Algoritmos y Estructuras de Datos I
Primer Cuatrimestre 2022

Guía Práctica 1
Lógica

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

1. Lógica binaria (Verdadero o Falso)

Ejercicio 1. ★ Sean p y q variables proposicionales. ¿Cuáles de las siguientes expresiones son *fórmulas bien formadas*?

a) $(p \neg q)$
b) $p \vee q \wedge True$
c) $(p \to \neg p \to q)$
d) $\neg(p)$
e) $(p \vee \neg p \wedge q)$
f) $(True \wedge True \wedge True)$
g) $(\neg p)$
h) $(p \vee False)$
i) $(p = q)$

Ejercicio 2. ★ Sean $x : \mathbb{Z}$, $y : \mathbb{Z}$ y $z : \mathsf{Bool}$ tres variables. Indique cuáles de las siguientes expresiones están bien definidas, teniendo en cuenta que estén bien tipadas las subexpresiones que correspondan.

a) $(1 = 0) \vee (x = y)$
b) $(x + 10) = y$
c) $(x \vee y)$
d) $z = \text{true} \leftrightarrow (y = x)$
e) $(z = 0) \vee (z = 1)$
f) $y + (y < 0)$

Ejercicio 3. La fórmula $(3 + 7 = \pi - 8) \wedge True$ es una fórmula bien formada. ¿Por qué? Justifique informal, pero detalladamente, su respuesta.

Ejercicio 4. ★ Determinar el valor de verdad de las siguientes proposiciones

a) $(\neg a \vee b)$
b) $(c \vee (y \wedge x) \vee b)$
c) $\neg(c \vee y)$
d) $(\neg(c \vee y) \leftrightarrow (\neg c \wedge \neg y))$
e) $((c \vee y) \wedge (x \vee b))$
f) $(((c \vee y) \wedge (x \vee b)) \leftrightarrow (c \vee (y \wedge x) \vee b))$
g) $(\neg c \wedge \neg y)$

cuando el valor de verdad de a, b y c es *verdadero*, mientras que el de x e y es *falso*.

Ejercicio 5. Determinar, utilizando tablas de verdad, si las siguientes fórmulas son tautologías, contradicciones o contingencias.

a) $(p \vee \neg p)$
b) $(p \wedge \neg p)$
c) $((\neg p \vee q) \leftrightarrow (p \to q))$
d) $((p \vee q) \to p)$
e) $(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q))$
f) $(p \to p)$
g) $((p \wedge q) \to p)$
h) $\big((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r))\big)$
i) $\big((p \to (q \to r)) \to ((p \to q) \to (p \to r))\big)$

Ejercicio 6. ★ Dadas las proposiciones lógicas α y β, se dice que α es más fuerte que β si y sólo si $\alpha \to \beta$ es una tautología. En este caso, también decimos que β es más débil que α. Determinar la relación de fuerza de los siguientes pares de fórmulas:

a) $True$, $False$
b) $(p \wedge q)$, $(p \vee q)$
c) $True$, $True$
d) p, $(p \wedge q)$
e) $False$, $False$
f) p, $(p \vee q)$
g) p, q
h) p, $(p \to q)$

¿Cuál es la proposición más fuerte y cuál la más débil de las que aparecen en este ejercicio?

Ejercicio 7. ★ Usando reglas de equivalencia (conmutatividad, asociatividad, De Morgan, etc) determinar si los siguientes pares de fórmulas son equivalencias. Indicar en cada paso qué regla se utilizó.

a)
- $((\neg p \vee \neg q) \vee (p \wedge q)) \rightarrow (p \wedge q)$
- $(p \wedge q)$

b)
- $(p \vee q) \wedge (p \vee r)$
- $\neg p \rightarrow (q \wedge r)$

c)
- $\neg(\neg p) \rightarrow (\neg(\neg p \wedge \neg q))$
- q

d)
- $((True \wedge p) \wedge (\neg p \vee False)) \rightarrow \neg(\neg p \vee q)$
- $p \wedge \neg q$

e)
- $(p \vee (\neg p \wedge q))$
- $\neg p \rightarrow q$

f)
- $\neg(p \wedge (q \wedge s))$
- $s \rightarrow (\neg p \vee \neg q)$

g)
- $p \rightarrow (q \wedge \neg(q \rightarrow r))$
- $(\neg p \vee q) \wedge (\neg p \vee (q \wedge \neg r))$

Ejercicio 8. Decimos que un conectivo es *expresable* mediante otros si es posible escribir una fórmula utilizando exclusivamente estos últimos y que tenga la misma tabla de verdad que el primero (es decir, son equivalentes). Por ejemplo, la disyunción es expresable mediante la conjunción más la negación, ya que $(p \vee q)$ tiene la misma tabla de verdad que $\neg(\neg p \wedge \neg q)$.

Mostrar que cualquier fórmula de la lógica proposicional que utilice los conectivos $\neg$ (negación), $\wedge$ (conjunción), $\vee$ (disyunción), $\rightarrow$ (implicación), $\leftrightarrow$ (equivalencia) puede reescribirse utilizando sólo los conectivos $\neg$ y $\vee$.

Ejercicio 9. ★ Sean las variables proposicionales f, e y m con los siguientes significados:

$f \equiv$"es fin de semana" $\quad e \equiv$"Juan estudia" $\quad m \equiv$"Juan escucha música"

a) Escribir usando lógica proposicional las siguientes oraciones:
- "Si es fin de semana, Juan estudia o escucha música, pero no ambas cosas"
- "Si no es fin de semana entonces Juan no estudia"
- "Cuando Juan estudia los fines de semana, lo hace escuchando música"

b) Asumiendo que valen las tres proposiciones anteriores ¿se puede deducir que Juan no estudia? Justificar usando argumentos de la lógica proposicional.

Ejercicio 10. En la salita verde de un jardín se sabe que las siguientes circunstancias son ciertas:

a) Si todos conocen a Juan entonces todos conocen a Camila (podemos pensar que esto se debe a que siempre caminan juntos).

b) Si todos conocen a Juan, entonces que todos conozcan a Camila implica que todos conocen a Gonzalo.

La pregunta entonces es: ¿Es cierto que si todos conocen a Juan entonces todos conocen a Gonzalo? Justificar.

Ejercicio 11. Siempre que Haroldo se pelea con sus compañeritos, vuelve a casa con un ojo morado. Si un día lo viéramos llegar con el ojo destrozado, podríamos sentirnos inclinados a concluir que se ha tomado a golpes de puño y cabezazos con los otros niñitos. ¿Puede identificar el error en el razonamiento anterior? *Pista:* Es conocido como *falacia de afirmar el consecuente.*

2. Lógica ternaria (Verdadero, Falso o Indefinido)

Ejercicio 12. ★ Asignar un *valor de verdad* (*verdadero*, *falso* o *indefinido*) a cada una de las siguientes *expresiones aritméticas* en los reales.

a) $5 > 0$

b) $1 \leq 1$

c) $(5 + 3 - 8)^{-1} \neq 2$

d) $0 \geq 5$

e) $\frac{1}{0} = \frac{1}{0}$

f) $0 > \log_2(2^{2^0 - 1} - 1)$

g) $0 \cdot \sqrt{-1} = 0$

h) $\sqrt{-1} \cdot 0 = 0$

i) $\tan(\frac{\pi}{2}) = \tan(\pi) - \tan(2)$

Ejercicio 13. ★ ¿Cuál es la diferencia entre el operador $\wedge$ y el operador $\wedge_L$? Describir la tabla de verdad de ambos operadores.

Ejercicio 14. ★ ¿Cuál es la diferencia entre el operador $\vee$ y el operador $\vee_L$? Describir la tabla de verdad de ambos operadores.

Ejercicio 15. ★ ¿Cuál es la diferencia entre el operador $\rightarrow$ y el operador $\longrightarrow_L$? Describir la tabla de verdad de ambos operadores.

Ejercicio 16. ★ Determinar los valores de verdad de las siguientes proposiciones cuando el valor de verdad de b y c es *verdadero*, el de a es *falso* y el de x e y es *indefinido*:

a) $(\neg x \vee_L b)$

b) $((c \vee_L (y \wedge_L a)) \vee b)$

c) $\neg(c \vee_L y)$

d) $(\neg(c \vee_L y) \leftrightarrow (\neg c \wedge_L \neg y))$

e) $((c \vee_L y) \wedge_L (a \vee_L b))$

f) $(((c \vee_L y) \wedge_L (a \vee_L b)) \leftrightarrow (c \vee_L (y \wedge_L a) \vee_L b))$

g) $(\neg c \wedge_L \neg y)$

Ejercicio 17. Sean p, q y r tres variables de las que se sabe que:

- p y q nunca están indefinidas,
- r se indefine sii q es *verdadera*

Proponer una fórmula que nunca se indefina, utilizando siempre las tres variables y que sea verdadera si y solo si se cumple que:

a) Al menos una es verdadera.

b) Ninguna es verdadera.

c) Exactamente una de las tres es verdadera.

d) Sólo p y q son verdaderas.

e) No todas al mismo tiempo son verdaderas.

f) r es verdadera.

3. Cuantificadores

Ejercicio 18.

a) ★ Determinar para cada aparición de variables, si dicha aparición se encuentra libre o ligada. En caso de estar ligada, aclarar a qué cuantificador lo está.

i) $(\forall x : \mathbb{Z})(0 \leq x < n \rightarrow\ x + y = z)$

ii) $(\forall x : \mathbb{Z})((\forall y : \mathbb{Z})((0 \leq x < n \wedge 0 \leq y < m) \rightarrow x + y = z))$

iii) $(\forall j : \mathbb{Z})(0 \leq j < 10 \rightarrow j < 0)$

iv) $s \wedge a < b - 1 \wedge ((\forall j : \mathbb{Z})(a \leq j < b \rightarrow_L 2 * j < b \vee s))$

v) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow (\forall j : \mathbb{Z})(j > 0 \rightarrow j \neq 0))$

vi) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow P(j))$

vii) $(\forall j : \mathbb{Z})(j \leq 0 \rightarrow P(j)) \wedge P(j)$

b) ★ En los casos en que sea posible, proponer valores para las variables libres del item anterior de modo tal que las expresiones sean verdaderas.

Ejercicio 19. Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen. Considerar los siguientes enunciados y su predicado asociado. Determinar, en cada caso, por qué el predicado no refleja correctamente el enunciado. Corregir los errores.

a) *"Todos los naturales menores a 10 que cumplen P, cumplen Q":*
pred $a(){(\forall x : \mathbb{Z})((0 \leq x < 10) \rightarrow_L (P(x) \wedge Q(x)))}$

b) *"No hay ningún natural menor a 10 que cumpla P y Q":*
pred $c(){\neg((\exists x : \mathbb{Z})(0 \leq x < 10 \wedge P(x) \wedge \neg((\exists x : \mathbb{Z})(0 \leq x < 10 \wedge Q(x)))))}$

4. Funciones auxiliares

Ejercicio 20. ★ Escriba los siguientes predicados y funciones en el lenguaje de especificación:

a) *aux suc* $(x : \mathbb{Z})$*:* $\mathbb{Z}$, que corresponde al sucesor de x.

b) *aux suma* $(x, y : \mathbb{R})$*:* $\mathbb{R}$, que corresponda a la suma entre x e y.

c) *aux producto* $(x, y : \mathbb{R})$*:* $\mathbb{R}$, que corresponde al producto entre x e y.

d) *pred esCuadrado* $(x : \mathbb{Z})$ que sea verdadero si y solo si x es un numero cuadrado.

e) *pred esPrimo* $(x : \mathbb{Z})$ que sea verdadero sii x es primo.

f) *pred sonCoprimos* $(x, y : \mathbb{Z})$ que sea verdadero si y solo si x e y son coprimos.

g) *pred divisoresGrandes* $(x, y : \mathbb{Z})$ que sea verdadero cuando todos los divisores de x, sin contar el uno, son mayores que y.

h) *pred mayorPrimoQueDivide* $(x : \mathbb{Z}, y : \mathbb{Z})$ que sea verdadero si y es el mayor primo que divide a x.

i) *pred sonPrimosHermanos* $(x : \mathbb{Z}, y : \mathbb{Z})$ que sea verdadero cuando x es primo, y es primo, y son primos consecutivos.

Algoritmos y Estructuras de Datos I
Primer Cuatrimestre 2022

Guía Práctica 2
Introducción al Lenguaje de Especificación

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

1. Secuencias

Ejercicio 1. ★ Evaluar las siguientes expresiones:

a) $|\langle 4, 3, 1\rangle|$

b) $\mathsf{addFirst}(\pi, \langle 2, 3, 5, 7, 11\rangle)$

c) $\langle 0, 1, 2, 3\rangle[3]$

d) $\mathsf{concat}(\langle 2, 3\rangle, \langle 5, 7, 11\rangle)$

e) $\mathsf{head}(\mathsf{tail}(\langle 5, 6, 7, 8\rangle))$

f) $\mathsf{subseq}(\langle 2, 3, 5, 7, 11\rangle, 0, 3)$

g) $\pi \in \langle 2, 3, 5, 7, 11\rangle$

h) $\mathsf{subseq}(\langle 2, 3, 5, 7, 11\rangle, 3, 2)$

i) $1 \in \langle 1, 2, 3, 4, 5\rangle$

j) $\mathsf{subseq}(\langle 2, 3, 5, 7, 11\rangle, 0, 65536)$

Ejercicio 2. ★ Sea x de tipo seq$\langle\mathbb{Z}\rangle$. ¿Cuáles de las siguientes igualdades sobre secuencias son válidas?

a) $|x| = |\mathsf{tail}(x)| + 1$

b) $x = \mathsf{subseq}(x, 0, |x| - 1)$

c) $x = \mathsf{subseq}(x, 0, |x|)$

d) $\mathsf{concat}(\mathsf{addFirst}(3, x), y) = \mathsf{addFirst}(3, \mathsf{concat}(x, y))$

e) $x = \mathsf{addFirst}(\mathsf{head}(x), \mathsf{tail}(x))$

f) $x[0] = \mathsf{head}(x)$

g) $i \in x = \mathsf{head}(\mathsf{subseq}(x, i, i + 1))$

h) $\mathsf{tail}(x) = \mathsf{subseq}(x, 1, |x|)$

En los casos incorrectos, ¿puede dar condiciones sobre las listas en cuestión para que lo sean?

Ejercicio 3. ★ Sea s_0, s_1 secuencias de tipo T y e un elemento de tipo T. Indicar para cada una de las siguientes afirmaciones si son verdaderas o falsas. En caso de ser falsa, mostrar un contraejemplo.

a) $|\mathsf{addFirst}(e, s_0)| = 1 + |s_0|$

b) $|\mathsf{addFirst}(e, s_0)| = |\mathsf{tail}(s_0)|$

c) $|\mathsf{concat}(s_0, s_1)| = |s_0| + |s_1|$

d) $s_0 = \mathsf{tail}(\mathsf{addFirst}(e, s_0))$

e) $\mathsf{head}(\mathsf{addFirst}(e, s_0)) = e$

f) $\mathsf{addFirst}(e, s_0) = \mathsf{tail}(s_0)$

g) $\mathsf{head}(\mathsf{addFirst}(e, \mathsf{tail}(s_0))) = \mathsf{head}(\mathsf{tail}(\mathsf{addFirst}(e, s_0)))$

h) $\mathsf{addFirst}(e, s_0)[0] = e$

i) $\mathsf{addFirst}(e, s_0)[0] = \mathsf{head}(\mathsf{addFirst}(e, s_0))$

Ejercicio 4. ★ Escriba los siguientes predicados auxiliares sobre secuencias de enteros, aclarando los tipos de los parámetros que recibe:

a) *estáAcotada*, que determina si todos los elementos de una secuencia están dentro del rango $[1, 100]$.

b) *capicúa*, que es verdadera sii una secuencia es capicúa. (Por ejemplo, $\langle 0, 2, 1, 2, 0\rangle$ es capicúa y $\langle 0, 2, 1, 4, 0\rangle$ no).

c) *esPrefijo*, que es verdadera sii una secuencia es prefijo de otra.

d) *estáOrdenada*, que es verdadera sii la secuencia está ordenada de menor a mayor.

e) *todosPrimos*, que es verdadera sii todos los elementos de la secuencia son números primos.

f) *primosEnPosicionesPares*, que es verdadero sii todos los elementos primos de una secuencia están en una posición par.

g) *todosIguales*, que es verdadera sii todos los elementos de la secuencia son iguales.

h) *hayUnoParQueDivideAlResto*, que determina si hay un elemento par en la secuencia que divide a todos los otros elementos de la secuencia.

i) *hayUnoEnPosiciónParQueDivideAlResto*, que determina si hay un elemento en una posición par de la secuencia que divide a todos los otros elementos contenidos en la secuencia.

j) *sinRepetidos*, que determina si la secuencia no tiene repetidos.

k) *otroMayorADerecha*, que determina si todo elemento de la secuencia, salvo el último, tiene otro mayor a su derecha.

l) *todoEsMúltiplo*, que determina si todo elemento de la secuencia es múltiplo de algún otro.

m) *enTresPartes*, que determina si en la secuencia aparecen (de izquierda a derecha) primero 0s, después 1s y por último 2s. Por ejemplo $\langle 0, 0, 1, 1, 1, 1, 2\rangle$ cumple con *enTresPartes*, pero $\langle 0, 1, 3, 0\rangle$ o $\langle 0, 0, 0, 1, 1\rangle$ no. ¿Cómo modificaría la expresión para que se admitan cero apariciones de 0s, 1s y 2s (es decir, para que por ejemplo $\langle 0, 0, 0, 1, 1\rangle$ o $\langle\rangle$ sí cumplan *enTresPartes*)?

n) *esPermutaciónOrdenada*, que dadas dos secuencias s y t sea verdadero sii s es permutación de t y está ordenada.

Ejercicio 5. Especificar las siguientes funciones y predicados auxiliares. En caso de no ser posible, explicar las razones.

a) *aux intercambiarPrimeroPorUltimo(s : seq⟨ℤ⟩) : seq⟨ℤ⟩*. Que intercambia el último valor por el primero en una secuencia.

b) *pred esReverso(s : seq⟨ℤ⟩, t : seq⟨ℤ⟩)*. Que indica si la secuencia s es el reverso de la secuencia t.

c) *aux reverso(s : seq⟨ℤ⟩) : seq⟨ℤ⟩*. Que indica el reverso de una secuencia.

d) *aux agregarTresCeros(s : seq⟨ℤ⟩) : seq⟨ℤ⟩*. Que agrega 3 ceros al final de la secuencia s.

e) *aux agregarNCeros(s : seq⟨ℤ⟩, n : ℤ) : seq⟨ℤ⟩*. Que agrega n ceros al final de la secuencia s.

f) *aux sumarUno(s : seq⟨ℤ⟩) : seq⟨ℤ⟩*. Que suma 1 a cada uno de los elementos de la secuencia s.

g) *aux ordenar(s : seq⟨ℤ⟩) : seq⟨ℤ⟩*. Que ordena la lista de menor a mayor.

Ejercicio 6. ★ Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen y sea s una secuencia de enteros. Escribir el predicado asociado a cada uno de los siguientes enunciados:

a) *"Si un entero en s cumple P, también cumple Q"*

b) *"Todos los enteros de s que cumplen P, no cumplen Q"*

c) *"Todos los enteros de s que están en posiciones pares y cumplen P, no cumplen Q"*

d) *"Todos los enteros de s que cumplen P y están en posiciones que cumplen Q, son pares"*

e) *"Si hay un entero en s que no cumple P entonces ninguno en s cumple Q"*

f) *"Si hay un entero en s que no cumple P entonces ninguno en s cumple Q; y si todos los enteros de s cumplen P entonces hay al menos dos elementos de s que cumplen Q"*

Ejercicio 7. ★ Sea $P(x : \mathbb{Z})$ un predicado cualquiera y s una secuencia de enteros. Explicar cuál es el error de traducción a fórmulas de los siguientes enunciados. Dar un ejemplo en el cuál sucede el problema y luego corregirlo.

a) *"Todo elemento en una posición válida de la secuencia cumple P"*: $(\forall i : \mathbb{Z})((0 \leq i < |s|) \wedge_L P(s[i]))$

b) *"Algún elemento en una posición válida de la secuencia cumple P"*: $(\exists i : \mathbb{Z})((0 \leq i < |s|) \rightarrow_L P(s[i]))$

Ejercicio 8. ★

Sean $P(x : \mathbb{Z})$ y $Q(x : \mathbb{Z})$ dos predicados cualesquiera que nunca se indefinen, sea s una secuencia de enteros y sean a, b y k enteros. Decidir en cada caso la relación de fuerza entre las dos fórmulas:

a) $P(3)$ y $(\forall k : \mathbb{Z})((0 \leq k < 10) \rightarrow P(k))$

b) $P(3)$ y $k > 5 \wedge (\forall i : \mathbb{Z})((0 \leq i < k) \rightarrow P(i)))$

c) $(\forall n : \mathbb{Z})((n \in s \wedge P(n)) \rightarrow Q(n))$ y $(\forall n : \mathbb{Z})((n \in s) \rightarrow Q(n))$

d) $(\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$ y $(\forall n : \mathbb{Z})((n \in s) \rightarrow Q(n))$

e) $(\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$ y $|s| > 0 \wedge ((\forall n : \mathbb{Z})((n \in s) \rightarrow Q(n)))$

f) $(\exists n : \mathbb{Z})(n \in s \wedge P(n) \wedge Q(n))$ y $(\forall n : \mathbb{Z})(n \in s \rightarrow (P(n) \wedge Q(n)))$

Ejercicio 9. Sea s una secuencia de enteros. Determinar si los siguientes pares de expresiones son equivalentes. En caso de que no lo sean, ilustrar con ejemplos.

a)
- $(\forall i : \mathbb{Z})((0 \leq i < |s|) \rightarrow_L ((\forall j : \mathbb{Z})(0 \leq j < |s|) \wedge i < j) \rightarrow_L s[i] < s[j])$ y
- $(\forall j : \mathbb{Z})((0 \leq j < |s|) \rightarrow_L ((\forall i : \mathbb{Z})(0 \leq i < |s|) \wedge i < j) \rightarrow_L s[i] < s[j])$.

b)
- $(\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L ((\exists j : \mathbb{Z})((0 \leq j < |s| \wedge i < j - 1) \wedge_L todosIguales(\mathsf{subseq}(s, i, j))))))$ y
- $(\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L ((\exists i : \mathbb{Z})((0 \leq i < |s| \wedge i < j - 1) \wedge_L todosIguales(\mathsf{subseq}(s, i, j))))))$.

donde *todosIguales* es el definido en el item *g)* del ejercicio *4*.

c)
- $(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L ((\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L \ s[i] = s[j]))$ y
- $(\exists j : \mathbb{Z})(0 \leq j < |s| \wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |s|) \rightarrow_L s[i] = s[j]))$.

2. Sumatorias y Productorias

Ejercicio 10. ★ Evaluar las siguientes expresiones:

a) $\sum_{i=0}^{2} \langle 4, 3, 1 \rangle[i]$

b) $\sum_{i=0}^{0} \langle \pi, 2, 3, 5, 7, 11 \rangle[i]$

c) $\sum_{i=0}^{-1} \langle 1, 2, 3, 4, 5 \rangle[i]$

d) $\sum_{i=0}^{5} \frac{1}{i}$

e) $\sum_{i=0}^{\sqrt{-1}} \langle 2, 3, 5, 7, 11 \rangle[i]$

f) $\sum_{i=15}^{2} \langle 2, 3, 5, 7, 11 \rangle[i]$

g) $\sum_{i=2}^{15} \langle 2, 3, 5, 7, 11 \rangle[i]$

h) $\sum_{i=1}^{3} \langle 2, 3, 5, 7, 11 \rangle[i]$

i) $\sum_{i=0}^{4} \langle 1, 1, 1, 1, 1 \rangle[i]$

j) $\sum_{i=0}^{4} \langle 0, 0, 0, 0, 0 \rangle[i]$

Ejercicio 11. ★ Escribir un predicado que usando sumatorias indique si un número entero es primo.

Ejercicio 12. Sea s una secuencia de elementos de tipo $\mathbb{Z}$. Escribir una expresión tal que:

a) Cuente la cantidad de veces que aparece el elemento e de tipo $\mathbb{Z}$ en la secuencia s.

b) Sume los elementos en las posiciones impares de la secuencia s.

c) Sume los elementos mayores a 0 contenidos en la secuencia s.

d) Sume los inversos multiplicativos ($\frac{1}{x}$) de los elementos contenidos en la secuencia s distintos a 0.

e) Cuente la cantidad de elementos primos no repetidos en la secuencia s.

Ejercicio 13. Escribir un predicado que indique si una secuencia es permutación de otra secuencia. Una secuencia es permutación de otra secuencia si ambas secuencias poseen los mismos elementos y la misma cantidad de apariciones por elemento. Ejemplos:

- $\langle 1, 2, 3 \rangle$ es permutación de $\langle 3, 2, 1 \rangle$.
- $\langle 1, 2, 3 \rangle$ es permutación de $\langle 1, 2, 3 \rangle$.
- $\langle 1, 1, 2, 3 \rangle$ es permutación de $\langle 3, 2, 1, 1 \rangle$.
- $\langle 1, 2, 3 \rangle$ no es permutación de $\langle 1, 1, 3 \rangle$.
- $\langle 1, 1, 2, 3 \rangle$ es permutación de $\langle 1, 3, 2, 1 \rangle$.

Ejercicio 14. ★ Sea m una secuencia de secuencias de tipo $\mathbb{Z}$, escribir una expresión tal que:

a) Sume los elementos contenidos en todas las secuencias.

b) Cuente la cantidad de secuencias vacías.

c) Sume el valor del último elemento de cada secuencia no vacía.

d) Retorne true sii todas las secuencias poseen el mismo tamaño.

e) Retorne la suma de todas las posiciones impares de cada secuencia.

Ejercicio 15. Sea s una $seq\langle \mathsf{Char} \rangle$, escribir una expresión que cuente la cantidad de apariciones del caracter vacío (' ').

Ejercicio 16. ★ Sea s una $seq\langle \mathsf{Char} \rangle$, escribir una expresión que cuente la cantidad de apariciones de dígitos (caracteres '0' al '9').

Algoritmos y Estructuras de Datos I
Primer Cuatrimestre 2022

Guía Práctica 3
Especificación de problemas

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1. ★ Las siguientes especificaciones no son correctas. Indicar por qué, y corregirlas para que describan correctamente el problema.

a) `buscar:` Dada una secuencia y un elemento de ésta, devuelve en *result* alguna posición de la secuencia en la cual se encuentre el elemento.

`proc buscar` (in l: $seq\langle\mathbb{R}\rangle$, in elem: $\mathbb{R}$, out result: $\mathbb{Z}$) {
`Pre` $\{elem \in l\}$
`Post` $\{l[result] = elem\}$
}

b) `progresionGeometricaFactor2:` Indica si la secuencia l representa una progresión geométrica factor 2. Es decir, si cada elemento de la secuencia es el doble del elemento anterior.

`proc progresionGeometricaFactor2` (in l: $seq\langle\mathbb{Z}\rangle$, out result: Bool) {
`Pre` $\{True\}$
`Post` $\{result = True \leftrightarrow ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] = 2 * l[i-1]))\}$
}

c) `minimo:` Devuelve en *result* el menor elemento de l.

`proc minimo` (in l: $seq\langle\mathbb{Z}\rangle$, out result: $\mathbb{Z}$) {
`Pre` $\{True\}$
`Post` $\{(\forall y : \mathbb{Z})((y \in l \wedge y \neq x) \rightarrow y > result)\}$
}

Ejercicio 2. La siguiente no es una especificación válida, ya que para ciertos valores de entrada que cumplen la precondición, no existe una salida que cumpla con la postcondición.

`proc elementosQueSumen` (in l: $seq\langle\mathbb{Z}\rangle$, in suma:$\mathbb{Z}$, out result: $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{True\}$
`Post` {
/* La secuencia result está incluída en la secuencia l*/
$(\forall x : \mathbb{Z})(x \in result \rightarrow \#apariciones(x, result) \leq \#apariciones(x, l))$
/* La suma de la result coincide con el valor suma */
$\wedge\ suma = \sum_{i=0}^{|result|-1} result[i]$
}
}

a) Mostrar valores para l y *suma* que hagan verdadera la precondición, pero tales que no exista *result* que cumpla la postcondición.

b) Supongamos que agregamos a la especificación la siguiente cláusula:

Pre : $min\_suma(l) \leq suma \leq max\_suma(l)$
fun $min\_suma(l) : \mathbb{Z} = \sum_{i=0}^{|l|-1}$ if $l[i] < 0$ then $l[i]$ else 0 fi
fun $max\_suma(l) : \mathbb{Z} = \sum_{i=0}^{|l|-1}$ if $l[i] > 0$ then $l[i]$ else 0 fi

¿Ahora es una especificación válida? Si no lo es, justificarlo con un ejemplo como en el punto anterior.

c) Dar una precondición que haga correcta la especificación.

Ejercicio 3. ★ Para los siguientes problemas, dar todas las soluciones posibles a las entradas dadas:

a) `proc raizCuadrada` (in x:$\mathbb{R}$, out result:$\mathbb{R}$) {

`Pre` $\{x \geq 0\}$

`Post` $\{result^2 = x\}$

}

i) $x = 0$

ii) $x = 1$

iii) $x = 27$

b) ★

`proc indiceDelMaximo` (in l: $seq\langle\mathbb{R}\rangle$, out result:$\mathbb{Z}$) {

`Pre` $\{|l| > 0\}$

`Post` {
$0 \leq result < |l|$
$\wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] \leq l[result])$
}

}

i) $l = \langle 1, 2, 3, 4\rangle$

ii) $l = \langle 15.5, -18, 4.215, 15.5, -1\rangle$

iii) $l = \langle 0, 0, 0, 0, 0, 0\rangle$

c) ★

`proc indiceDelPrimerMaximo` (in l:$seq\langle\mathbb{R}\rangle$, out result:$\mathbb{Z}$) {

`Pre` $\{|l| > 0\}$

`Post` {
$0 \leq result < |l|$
$\wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L (l[i] < l[result] \vee (l[i] = l[result] \wedge i \geq result))))$
}

}

i) $l = \langle 1, 2, 3, 4\rangle$

ii) $l = \langle 15.5, -18, 4.215, 15.5, -1\rangle$

iii) $l = \langle 0, 0, 0, 0, 0, 0\rangle$

d) ¿Para qué valores de entrada indiceDelPrimerMaximo y indiceDelMaximo tienen necesariamente la misma salida?

Ejercicio 4. ★ Sea $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ definida como:

$$f(a, b) = \begin{cases} 2b & \text{si } a < 0 \\ b - 1 & \text{en otro caso} \end{cases}$$

¿Cuáles de las siguientes especificaciones son correctas para el problema de calcular $f(a, b)$?
Para las que no lo son, indicar por qué.

a) `proc f` (in a, b: $\mathbb{R}$,out result:$\mathbb{R}$) {

`Pre` $\{True\}$

`Post` {
$(a < 0 \land result = 2 * b)$
$\land$
$(a \geq 0 \land result = b - 1)$
}

}

b) `proc f` (in a, b: $\mathbb{R}$, out result:$\mathbb{R}$) {

`Pre` $\{True\}$

`Post` $\{(a < 0 \land result = 2 * b) \lor (a > 0 \land result = b - 1)\}$

}

c) `proc f` (in a, b: $\mathbb{R}$, out result:$\mathbb{R}$) {

`Pre` $\{True\}$

`Post` $\{(a < 0 \land result = 2 * b) \lor (a \geq 0 \land result = b - 1)\}$

}

d) `proc f` (in a, b: $\mathbb{R}$, out result:$\mathbb{R}$) {

`Pre` $\{True\}$

`Post` {
$(a < 0 \rightarrow result = 2 * b)$
$\land$
$(a \geq 0 \rightarrow result = b - 1)$
}

}

e) `proc f` (in a, b: $\mathbb{R}$, out result:$\mathbb{R}$) {

`Pre` $\{True\}$

`Post` $\{(a < 0 \rightarrow result = 2 * b) \lor (a \geq 0 \rightarrow result = b - 1)\}$

}

f) `proc f` (in a, b: $\mathbb{R}$, out result: $\mathbb{R}$) {

`Pre` $\{True\}$

`Post` $\{result = (\text{if } a < 0 \text{ then } 2 * b \text{ else } b - 1 \text{ fi})\}$

}

Ejercicio 5. ★ Considerar la siguiente especificación, junto con un algoritmo que dado x devuelve x^2.

`proc unoMasGrande` (in x: $\mathbb{R}$, out result:$\mathbb{R}$) {
`Pre` $\{True\}$
`Post` $\{result > x\}$
}

a) ¿Qué devuelve el algoritmo si recibe $x = 3$? ¿El resultado hace verdadera la postcondición de unoMasGrande?

b) ¿Qué sucede para las entradas x = 0.5, x = 1, x = -0.2 y x = -7?

c) Teniendo en cuenta lo respondido en los puntos anteriores, escribir una **precondición** para unoMasGrande, de manera tal que el algoritmo cumpla con la especificación.

Ejercicio 6. ★ Sean x y r variables de tipo $\mathbb{R}$. Considerar los siguientes predicados:

P1: $\{x \leq 0\}$ Q1: $\{r \geq x^2\}$
P2: $\{x \leq 10\}$ Q2: $\{r \geq 0\}$
P3: $\{x \leq -10\}$ Q3: $\{r = x^2\}$

a) Indicar la relación de fuerza entre P1, P2 y P3.

b) Indicar la relación de fuerza enrte Q1, Q2 y Q3.

c) Escribir 2 programas que cumplan con la siguiente especificación E1:

```
proc hagoAlgo (in x: ℝ, out r:ℝ)  {
      Pre {x ≤ 0}
      Post {r ≥ x²}
}
```

d) Sea A un algoritmo que cumple con la especificación E1 del ítem anterior. Decidir si necesariamente cumple las siguientes especificaciones:

a) Pre: $\{x \leq -10\}$, Post: $\{r \geq x^2\}$
b) Pre: $\{x \leq 10\}$, Post: $\{r \geq x^2\}$
c) Pre: $\{x \leq 0\}$, Post: $\{r \geq 0\}$
d) Pre: $\{x \leq 0\}$, Post: $\{r = x^2\}$
e) Pre: $\{x \leq -10\}$, Post: $\{r \geq 0\}$
f) Pre: $\{x \leq 10\}$, Post: $\{r \geq 0\}$
g) Pre: $\{x \leq -10\}$, Post: $\{r = x^2\}$
h) Pre: $\{x \leq 10\}$, Post: $\{r = x^2\}$

e) ¿Qué conclusión pueden sacar? ¿Qué debe cumplirse con respecto a las precondiciones y postcondiciones para que sea seguro reemplazar la especificación?

Ejercicio 7. ★ Considerar las siguientes dos especificaciones, junto con un algoritmo a que satisface la especificación de `p2`.

```
proc p1 (in x: ℝ, in n: ℤ, out result:ℤ)  {
      Pre {x ≠ 0}
      Post {x^n − 1 < result ≤ x^n}
}

proc p2 (in x: ℝ, in n: ℤ, out result:ℤ)  {
      Pre {n ≤ 0 → x ≠ 0}
      Post {result = ⌊x^n⌋}
}
```

a) Dados valores de x y n que hacen verdadera la precondición de `p1`, demostrar que hacen también verdadera la precondición de $p2$.

b) Ahora, dados estos valores de x y n, supongamos que se ejecuta a: llegamos a un valor de res que hace verdadera la postcondición de `p2`. ¿Será también verdadera la postcondición de `p1` con este valor de res?

c) ¿Podemos concluir que a satisface la especificación de `p1`?

Ejercicio 8. Considerar las siguientes especificaciones:

```
proc n-esimo1 (in l: seq⟨ℤ⟩, in n: ℤ, out result: ℤ)  {
      Pre {
      /*Los elementos están ordenados*/
      (∀i : ℤ)(0 ≤ i < |l| − 1 →_L l[i] < l[i + 1])
```

$\land\ 0 \leq n < |l|$
}
Post $\{result = l[n]\}$
}

proc n-esimo2 (in l: $seq\langle\mathbb{Z}\rangle$, in n: $\mathbb{Z}$, out result:$\mathbb{Z}$) {
Pre {
/*Los elementos son distintos entre sí*/
$(\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L ((\forall j : \mathbb{Z})(0 \leq j < |l| \land i \neq j) \rightarrow_L l[i] \neq l[j]))$
$\land$
$0 \leq n < |l|$
}
Post {
$result \in l$
$\land$
$n = \sum_{i=0}^{|l|-1}$ (if $l[i] < result$ then 1 else 0 fi)
}
}

¿Es cierto que todo algoritmo que cumple con n-esimo1 cumple también con n-esimo2? ¿Y al revés?
Sugerencia: Razonar de manera análoga a la del ejercicio anterior.

Ejercicio 9. ★ Especificar los siguientes problemas:

a) Dado un número entero, decidir si es par.

b) Dado un entero n y uno m, decidir si n es un múltiplo de m.

c) Dado un número real, devolver su inverso multiplicativo.

d) Dada una secuencia de caracteres, obtener de ella sólo los que son numéricos (con todas sus apariciones sin importar el orden de aparición).

e) Dada una secuencia de reales, devolver la secuencia que resulta de duplicar sus valores en las posiciones impares

f) Dado un número entero, listar todos sus divisores positivos (sin duplicados).

Ejercicio 10. Considerar el problema de decidir, siendo n y m enteros, cuándo n es múltiplo de m, y la siguiente especificación.

proc esMultiplo (in n, m: $\mathbb{Z}$, out result:Bool) {
Pre $\{m \neq 0\}$
Post $\{result = (n \mod m = 0)\}$
}

a) Según la definición matemática de múltiplo, ¿tiene sentido preguntarse si 4 es múltiplo de 0? ¿Cúal es la respuesta?

b) ¿Debería ser $n = 4$, $m = 0$ una entrada válida para el problema? ¿Lo es en esta especificación?

c) Corregir la especificación de manera tal que $n = 4$, $m = 0$ satisfaga la precondición (¡cuidado con las indefiniciones!).

d) ¿Qué relación de fuerza hay entre la precondición nueva y la original?

Ejercicio 11. Considerar el problema de, dada una secuencia de números reales, devolver la que resulta de duplicar sus valores en las posiciones impares.

a) Para la secuencia $\langle 1, 2, 3, 4\rangle$, ¿es $\langle 0, 4, 0, 8\rangle$ un resultado correcto?

b) Sea la siguiente especificación:

proc duplicarEnImpares (in l: $seq\langle\mathbb{R}\rangle$, out result: $seq\langle\mathbb{R}\rangle$) {
Pre $\{True\}$
Post $\{|result| = |l| \land (\forall i : \mathbb{Z})((0 \leq i < |result| \land i \mod 2 = 1) \rightarrow_L result[i] = 2 * l[i])\}$
}

Si $l = \langle 1, 2, 3, 4\rangle$, ¿$result = \langle 0, 4, 0, 8\rangle$ satisface la postcondición?

c) Si es necesario, corregir la especificación para que describa correctamente el resultado esperado.

d) ¿Qué relación de fuerza hay entre la nueva postcondición y la original?

Ejercicio 12. ★ Especificar el problema de dado un entero positivo retornar una secuencia de 0s y 1s que represente ese número en base 2 (es decir, en binario).

Ejercicio 13. Con lo visto en los ejercicios 9 a 12, ¿Encuentra casos de sub y sobreespecificación en las especificaciones del ejercicio 8?

Ejercicio 14. Especificar los siguientes problemas:

a) ★ Dado un número entero positivo, obtener la suma de sus factores primos.

b) Dado un número entero positivo, decidir si es perfecto. Se dice que un número es perfecto cuando es igual a la suma de sus divisores propios (es decir, todos salvo sí mismo).

c) Dado un número entero positivo n, obtener el menor entero positivo $m > 1$ tal que m sea coprimo con n.

d) ★ Dado un entero positivo, obtener su descomposición en factores primos. Devolver una secuencia de tuplas (p, e), donde p es un factor primo y e es su exponente, ordenada en forma creciente con respecto a p.

e) Dada una secuencia de números reales, obtener la diferencia máxima entre dos de sus elementos.

f) ★ Dada una secuencia de números enteros, devolver aquel que divida a más elementos de dicha secuencia. El elemento tiene que pertenecer a la secuencia original. Si existe más de un elemento que cumple esta propiedad, devolver alguno de ellos.

Ejercicio 15. Especificar los siguientes problemas sobre secuencias:

a) proc nEsimaAparicion(in $l : seq\langle\mathbb{R}\rangle$, in $e : \mathbb{R}$, in $n : \mathbb{Z}$, out $result : \mathbb{Z}$), que devuelve el índice de la n-ésima aparición de e en l.

b) Dadas dos secuencias s y t, decidir si s es una subcadena de t.

c) ★ Dadas dos secuencias s y t, decidir si s está *incluida* en t, es decir, si todos los elementos de s aparecen en t en igual o mayor cantidad.

d) proc mezclarOrdenado(in $s, t : seq\langle\mathbb{Z}\rangle$, out $result : seq\langle\mathbb{Z}\rangle$), que recibe dos secuencias ordenadas y devuelve el resultado de intercalar sus elementos de manera ordenada.

e) Dadas dos secuencias s y t especificar el procedimiento *intersecciónSinRepetidos* que retorna una secuencia que contiene únicamente los elementos que aparecen en ambas secuencias.

f) ★ Dadas dos secuencias s y t, devolver su *intersección*, es decir, una secuencia con todos los elementos que aparecen en ambas. Si un mismo elemento tiene repetidos, la secuencia retornada debe contener la cantidad mínima de apariciones en de s y de t.

Ejercicio 16. Especificar los siguientes problemas:

a) proc cantApariciones(in $l : seq\langle\mathsf{Char}\rangle$, out $result : seq\langle\mathsf{Char} \times \mathbb{Z}\rangle$) que devuelve la secuencia con todos los elementos de l, sin duplicados, con su cantidad de apariciones (en un orden cualquiera). Ejemplos:

- $cantApariciones(\langle 'a'\rangle) = \langle\langle 'a', 1\rangle\rangle$
- $cantApariciones(\langle 'a', 'b', 'c'\rangle) = \langle\langle 'a', 1\rangle, \langle 'c', 1\rangle, \langle 'b', 1\rangle\rangle$
- $cantApariciones(\langle 'a', 'b', 'c', 'b', 'd', 'b'\rangle) = \langle\langle 'a', 1\rangle, \langle 'b', 3\rangle, \langle 'd', 1\rangle, \langle 'c', 1\rangle\rangle$
- $cantApariciones(\langle\rangle) = \langle\rangle$

b) Dada una secuencia, devolver una secuencia de secuencias que contenga todos sus prefijos, en orden creciente de longitud.

c) ★ Dada una secuencia de secuencias de enteros l, devolver una secuencia de l que contenga el máximo valor. Por ejemplo, si $l = \langle\langle 2, 3, 5\rangle, \langle 8, 1\rangle, \langle 2, 8, 4, 3\rangle\rangle$, devolver $\langle 8, 1\rangle$ o $\langle 2, 8, 4, 3\rangle$.

d) proc interseccionMultiple(in $ls : seq\langle seq\langle\mathbb{R}\rangle\rangle$, out $l : seq\langle\mathbb{R}\rangle$) que devuelve en l el resultado de la intersección de todas las secuencias de ls.

e) ★ Dada una secuencia l con todos sus elementos distintos, devolver la secuencia de *partes*, es decir, la secuencia de todas las secuencias incluidas en l, cada una con sus elementos en el mismo orden en que aparecen en l.

Especificación de problemas usando **inout**

Ejercicio 17. ★ Dados dos enteros a y b, se necesita calcular su suma, y retornarla en un entero c. ¿Cúales de las siguientes especificaciones son correctas para este problema? Para las que no lo son, indicar por qué.

a) `proc sumar` (inout a, b, c: $\mathbb{Z}$) {
   `Pre` $\{True\}$
   `Post` $\{a + b = c\}$
}

b) `proc sumar` (in a, b: $\mathbb{Z}$, in c: $\mathbb{Z}$) {
   `Pre` $\{True\}$
   `Post` $\{c = a + b\}$
}

c) `proc sumar` (in a, b: $\mathbb{Z}$, out c: $\mathbb{Z}$) {
   `Pre` $\{True\}$
   `Post` $\{c = a + b\}$
}

d) `proc sumar` (inout a, b: $\mathbb{Z}$, out c: $\mathbb{Z}$) {
   `Pre` $\{a = A_0 \land b = B_0\}$
   `Post` $\{a = A_0 \land b = B_0 \land c = a + b\}$
}

Ejercicio 18. ★ Dada una secuencia l, se desea sacar su primer elemento y devolverlo. Decidir cúales de estas especificaciones son correctas. Para las que no lo son, indicar por qué y justificar con ejemplos.

a) `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$, out result:$\mathbb{R}$) {
   `Pre` $\{|l| > 0\}$
   `Post` $\{result = \mathsf{head}(l)\}$
}

b) `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$, out result: $\mathbb{R}$) {
   `Pre` $\{|l| > 0 \land l = L_0\}$
   `Post` $\{result = \mathsf{head}(L_0)\}$
}

c) `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$, out result:$\mathbb{R}$) {
   `Pre` $\{|l| > 0\}$
   `Post` $\{result = \mathsf{head}(L_0) \land |l| = |L_0| - 1\}$
}

d) `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$, out result:$\mathbb{R}$) {
   `Pre` $\{|l| > 0 \land l = L_0\}$
   `Post` $\{result = \mathsf{head}(L_0) \land l = \mathsf{tail}(L_0)\}$
}

e) `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$, out result:$\mathbb{R}$) {

`Pre` $\{|l| > 0 \land l = L_0\}$

`Post` {
$result = \mathsf{head}(L_0)$
$\land\ |l| = |L_0| - 1$
$\land_L\ ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] = L_0[i+1]))$
}

}

Ejercicio 19. Considerar la siguiente especificación:

`proc intercambiar` (inout l: $seq\langle\mathbb{R}\rangle$, in i, j: $\mathbb{Z}$) {
`Pre` $\{0 \leq i < |l| \land 0 \leq j < |l| \land l = L_0\}$
`Post` {
/*Las secuencias tienen la misma longitud*/
$|l| = |L_0|$
$\land$
/*Intercambia i*/
$l[i] = L_0[j]$
$\land$
/*Intercambia j*/
$l[j] = L_0[i]$
}
}

a) ¿Esta especificación es válida? Si lo es, ¿qué problema describe?

b) Mostrar con un ejemplo que la postcondición está sub-especificada (es decir, que hay valores que la hacen verdadera aunque no son deseables como solución).

c) Corregir la especificación agregando una o más cláusulas a la postcondición.

Ejercicio 20. Explicar coloquialmente la siguiente especificación:

`proc copiarPrimero` (inout l:$seq\langle\mathbb{Z}\rangle$, inout i:$\mathbb{Z}$) {
`Pre` {
/*Valores iniciales*/
$l = L_0 \land i = I_0$
$\land$
/*Secuencia no vacía*/
$|l| > 0$
$\land$
/*Indice en rango*/
$0 \leq i < |l|$
}
`Post` {
$|l| = |L_0|$
$\land_L$
$l[I_0] = L_0[0]$
$\land$
$i = L_0[I_0]$
$\land$
$((\forall j : \mathbb{Z})((0 \leq j < |l| \land j \neq I_0) \rightarrow_L l[j] = L_0[I_0])$
}
}

Ejercicio 21. Dada una secuencia de enteros, se requiere multiplicar por 2 aquéllos valores que se encuentran en posiciones pares. Indicar por qué son incorrectas las siguientes especificaciones, y proponer una alternativa correcta.

a) `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$) {

`Pre` $\{l = L_0\}$

`Post` {
$|l| = |L_0|$
$\wedge$
$(\forall i : \mathbb{Z})(0 \leq i < |l| \wedge i \mod 2 = 0) \rightarrow_L l[i] = 2 * L_0[i]$
}

}

b) `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$) {

`Pre` $\{l = L_0\}$

`Post` $\{(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \mod 2 \neq 0) \rightarrow_L l[i] = L_0[i])$
$\wedge$
$(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \mod 2 = 0) \rightarrow_L l[i] = 2 * L_0[i])$
}

}

c) `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$, out result:$seq\langle\mathbb{Z}\rangle$) {

`Pre` $\{True\}$

`Post` $\{|l| = |result|$
$\wedge$
$(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \mod 2 \neq 0) \rightarrow_L result[i] = l[i])$
$\wedge$
$(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge i \mod 2 = 0) \rightarrow_L result[i] = 2 * l[i])$
}

}

Ejercicio 22. Especificar los siguientes problemas de modificación de secuencias:

a) ★ proc primosHermanos(inout $l : seq\langle\mathbb{Z}\rangle$), que dada una secuencia de enteros mayores a dos, reemplaza dichos valores por el número primo menor más cercano. Por ejemplo, si $l = \langle 6, 5, 9, 14\rangle$, luego de aplicar primosHermanos(l), $l = \langle 5, 3, 7, 13\rangle$.

b) ★ proc reemplazar(inout $l : seq\langle\mathsf{Char}\rangle$, in $a, b : \mathsf{Char}$), que reemplaza todas las apariciones de a en l por b.

c) proc recortar(inout $l : seq\langle\mathbb{Z}\rangle$, in $a : \mathbb{Z}$), que saca de l todas las apariciones de a consecutivas que aparezcan al principio. Por ejemplo recortar($\langle 2, 2, 3, 2, 4\rangle, 2$) = $\langle 3, 2, 4\rangle$, mientras que recortar($\langle 2, 2, 3, 2, 4\rangle, 3$) = $\langle 2, 2, 3, 2, 4\rangle$.

d) proc intercambiarParesConImpares(inout $l : seq\langle\mathsf{Char}\rangle$), que toma una secuencia de longitud par y la modifica de modo tal que todas las posiciones de la forma $2k$ quedan intercambiadas con las posiciones $2k + 1$. Por ejemplo, intercambiarParesConImpares("*adinle*") modifica de la siguiente manera: "*daniel*".

e) ★proc limpiarDuplicados(inout $l : seq\langle\mathsf{Char}\rangle$, out $dup : seq\langle\mathsf{Char}\rangle$), que elimina los elementos duplicados de l dejando sólo su primera aparición (en el orden original). Devuelve además, *dup* una secuencia con todas las apariciones eliminadas (en cualquier orden).

Algoritmos y Estructuras de Datos I
Primer Cuatrimestre 2022

Guía Práctica 4
Precondición más débil en SmallLang

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1. ★ Calcular las siguientes expresiones, donde a, b son variables reales, i una variable entera y A es una secuencia de reales.

a) $\text{def}(a + 1)$.

b) $\text{def}(a/b)$.

c) $\text{def}(\sqrt{a/b})$.

d) $\text{def}(A[i] + 1)$.

e) $\text{def}(A[i + 2])$.

f) $\text{def}(0 \leq i \leq |A|)$.

g) $\text{def}(0 \leq i \leq |A| \ \wedge_L \ A[i] \geq 0)$.

Ejercicio 2. Calcular las siguientes precondiciones más débiles, donde a, b son variables reales, i una variable entera y A es una secuencia de reales.

a) $wp(\textbf{a := a+1}, a \geq 0)$.

b) $wp(\textbf{a := a/b}, a \geq 0)$.

c) $wp(\textbf{a := A[i]}, a \geq 0)$.

d) $wp(\textbf{a := b*b}, a \geq 0)$.

e) $wp(\textbf{b := b+1}, a \geq 0)$.

Ejercicio 3. ★ Calcular las siguientes precondiciones más débiles, donde a, b son variables reales, i una variable entera y A es una secuencia de reales.

a) $wp(\textbf{a := a+1; b := a/2}, b \geq 0)$.

b) $wp(\textbf{a := A[i] + 1; b := a*a}, b \neq 2)$.

c) $wp(\textbf{a := A[i] + 1; a := b*b}, a \geq 0)$.

d) $wp(\textbf{a := a-b; b := a+b}, a \geq 0 \wedge b \geq 0)$.

Ejercicio 4. ★ Sea $Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \geq 0)$. Calcular las siguientes precondiciones más débiles, donde i es una variable entera y A es una secuencia de reales.

a) $wp(\textbf{A[i] := 0}, Q)$.

b) $wp(\textbf{A[i+2] := 0}, Q)$.

c) $wp(\textbf{A[i+2] := -1}, Q)$.

d) $wp(\textbf{A[i] := 2 * A[i]}, Q)$.

e) $wp(\textbf{A[i] := A[i-1]}, Q)$.

Ejercicio 5. Calcular $wp(S, Q)$, para los siguientes pares de programas S y postcondiciones Q.

a) $S \equiv$ `i := i + 1`
$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \neq 0)$

b) $S \equiv$ `A[0] := 4`
$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \neq 0)$

c) $S \equiv$ `A[2] := 4`
$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |A| \rightarrow_L A[j] \neq 0)$

d) $S \equiv$ `A[i] := A[i+1] - 1`
$Q \equiv (\forall j : \mathbb{Z})(0 < j < |A| \rightarrow_L A[j] \geq A[j-1])$

e) $S \equiv$ `A[i] := A[i+1] - 1`
$Q \equiv (\forall j : \mathbb{Z})(0 < j < |A| \rightarrow_L A[j] \leq A[j-1])$

Ejercicio 6. Escribir programas para los siguientes problemas y demostrar formalmente su corrección usando la precondición más débil.

a) `proc problema1` (inout a: $\mathbb{Z}$) {
`Pre` $\{a = a_0 \ \wedge \ a \geq 0\}$
`Post` $\{a = a_0 + 2\}$
}

b) `proc problema2` (in a: $\mathbb{Z}$, out b: $\mathbb{Z}$) {
`Pre` $\{a \neq 0\}$
`Post` $\{b = a + 3\}$
}

c) `proc problema3` (in a: $\mathbb{Z}$, in b: $\mathbb{Z}$, out c: $\mathbb{Z}$) {
`Pre` $\{\text{true}\}$
`Post` $\{c = a + b\}$
}

d) `proc problema4` (in a: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
`Pre` $\{0 \leq i < |a|\}$
`Post` $\{result = 2 * a[i]\}$
}

e) `proc problema5` (in a: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
`Pre` $\{0 \leq i \ \wedge \ i + 1 < |a|\}$
`Post` $\{result = a[i] + a[i+1]\}$
}

Ejercicio 7. ★ Calcular $wp(S, Q)$, para los siguientes pares de programas S y postcondiciones Q.

a) $S \equiv$

```
if( a < 0 )
  b := a
else
  b := −a
endif
```

$Q \equiv (b = -|a|)$

b) $S \equiv$

```
if( a < 0 )
  b := a
else
  b := −a
endif
```

$Q \equiv (b = |a|)$

c) $S \equiv$

```
if( i > 0 )
  s[i] := 0
else
  s[0] := 0
endif
```

$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \geq 0)$

d) $S \equiv$

```
if( i > 1 )
  s[i] := s[i−1]
else
  s[i] := 0
endif
```

$Q \equiv (\forall j : \mathbb{Z})(1 \leq j < |s| \rightarrow_L s[j] = s[j-1])$

e) $S \equiv$

```
if( s[i] < 0 )
  s[i] := −s[i]
else
  skip
endif
```

$Q \equiv 0 \leq i < |s| \ \wedge_L \ s[i] \geq 0$

f) $S \equiv$

```
if( s[i] > 0 )
  s[i] := −s[i]
else
  skip
endif
```

$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |s| \ \rightarrow_L \ s[j] \geq 0)$

Ejercicio 8. ★ Escribir programas para los siguientes problemas y demostrar formalmente su corrección usando la precondición más débil.

a) `proc problema1` (in s: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, inout a: $\mathbb{Z}$) {

`Pre` $\{0 \leq i < |s| \ \wedge_L \ a = \sum_{j=0}^{i-1} s[j]\}$

`Post` $\{a = \sum_{j=0}^{i} s[j]\}$

}

b) `proc problema2` (in s: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, inout a: $\mathbb{Z}$) {

`Pre` $\{0 \leq i < |s| \ \wedge_L \ a = \sum_{j=0}^{i} s[j]\}$

`Post` $\{a = \sum_{j=1}^{i} s[j]\}$

}

c) `proc problema3` (in s: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, out res: Bool) {

`Pre` $\{0 \leq i < |s| \ \wedge_L \ (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] \geq 0)\}$

`Post` $\{res = true \leftrightarrow (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] \geq 0)\}$

}

d) `proc problema4` (in s: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, inout a: $\mathbb{Z}$) {

`Pre` $\{0 \leq i < |s| \ \wedge_L \ a = \sum_{j=0}^{i-1}(\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

`Post` $\{a = \sum_{j=0}^{i}(\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

}

e) `proc problema5` (in s: $seq\langle\mathbb{Z}\rangle$, in i: $\mathbb{Z}$, inout a: $\mathbb{Z}$) {

`Pre` $\{0 < i \leq |s| \ \wedge_L \ a = \sum_{j=1}^{i-1}(\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

`Post` $\{a = \sum_{j=0}^{i-1}(\text{if } s[j] \neq 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$

}

Algoritmos y Estructuras de Datos I
Primer Cuatrimestre 2022

Guía Práctica 5
Demostración de corrección de ciclos en SmallLang

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Teorema del invariante: corrección de ciclos

Ejercicio 1. ★ Consideremos el problema de sumar los elementos de un arreglo y la siguiente implementación en SmallLang, con el invariante del ciclo.

Especificación

```
proc sumar (in s: seq⟨ℤ⟩, out result: ℤ) {
    Pre {True}
    Post {result = ∑_{j=0}^{|s|-1} s[j]}
}
```

Implementación en SmallLang

```
result := 0;
i := 0;
while (i < s.size()) do
  result := result + s[i];
  i := i + 1
endwhile
```

Invariante de Ciclo

$$I \equiv 0 \leq i \leq |s| \wedge_L result = \sum_{j=0}^{i-1} s[j]$$

a) Escribir la precondición y la postcondición del ciclo.

b) ¿Qué punto falla en la demostración de corrección si el primer término del invariante se reemplaza por $0 \leq i < |s|$?

c) ¿Qué punto falla en la demostración de corrección si el límite superior de la sumatoria $(i-1)$ se reemplaza por i?

d) ¿Qué punto falla en la demostración de corrección si se invierte el orden de las dos instrucciones del cuerpo del ciclo?

e) Demostrar formalmente la corrección parcial del ciclo, usando los primeros puntos del teorema del invariante.

f) Proponer una función variante y demostrar formalmente la terminación del ciclo, utilizando la función variante.

Ejercicio 2. ★ Dadas la especificación y la implementación del problema `sumarParesHastaN`, escribir la precondición y la postcondición del ciclo, y demostrar formalmente su corrección a través del teorema del invariante.

Especificación

```
proc sumarParesHastaN (in n: ℤ, out result: ℤ) {
    Pre {n ≥ 0}
    Post {result = ∑_{j=0}^{n-1}(if j mod 2 = 0 then j else 0 fi)}
}
```

Implementación en SmallLang

```
result := 0;
i := 0;
while (i < n) do
  result := result + i;
  i := i + 2
endwhile
```

Invariante de ciclo

$$I \equiv 0 \leq i \leq n+1 \wedge i \bmod 2 \ = \ 0 \wedge result = \sum_{j=0}^{i-1} (\text{if } j \bmod 2 = 0 \text{ then } j \text{ else } 0 \text{ fi})$$

Ejercicio 3. Supongamos que se desea implementar la función `exponenciacion`, cuya especificación es la siguiente:

`proc exponenciacion` (in m: $\mathbb{Z}$, in n: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
  Pre $\{n \geq 0 \land \neg(m = 0 \land n = 0)\}$
  Post $\{result = m^n\}$
}

Consideremos además el siguiente invariante: $I \equiv 0 \leq i \leq n \land result = m^i$

a) Escribir un programa en SmallLang que resuelva este problema, y que incluya un ciclo que tenga a I como invariante. Demostrar formalmente la corrección de este ciclo.

b) La siguiente implementación en SmallLang es trivialmente errónea. ¿Qué punto del teorema del invariante falla en este caso?[1]

```
i := 0;
result := 0;

while( i < m ) do
  result := result * n;
  i := i + 1
endwhile
```

c) ¿Qué puede decir de la siguiente implementación en SmallLang? En caso de que sea correcta, proporcione una demostración. En caso de que sea incorrecta, explique qué punto del teorema del invariante falla.

```
i := 0;
result := 1;
while( i < n ) do
  i := i + 1;
  result := result * m
endwhile
```

d) ¿Qué puede decir de la siguiente implementación? En caso de que sea incorrecta, ¿se puede reforzar la precondición del problema para que esta especificación pase a ser correcta?

```
i := 2;
result := m*m;

while( i < n ) do
  result := result * m;
  i := i + 1
endwhile
```

Ejercicio 4. ★ Considere el problema `sumaDivisores`, dado por la siguiente especificación:

`proc sumaDivisores` (in n: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
  Pre $\{n \geq 1\}$
  Post $\{result = \sum_{j=1}^{n}(\text{if } n \ mod \ j = 0 \text{ then } j \text{ else } 0 \text{ fi})\}$
}

a) Escribir un programa en SmallLang que satisfaga la especificación del problema y que contenga exactamente un ciclo.

b) El ciclo del programa propuesto, ¿puede ser demostrado mediante el siguiente invariante?

$$I \equiv 1 \leq i \leq n \land result = \sum_{j=1}^{i}(\text{if } n \ mod \ j = 0 \text{ then } j \text{ else } 0 \text{ fi})$$

Si no puede, ¿qué cambios se le deben hacer al invariante para que se corresponda con el ciclo propuesto?

[1] Recordar que para mostrar que una implicación $A \rightarrow B$ no es cierta alcanza con dar valores de las variables libres que hagan que A sea verdadero y que B sea falso. Para mostrar que una tripla de Hoare $\{P\}S\{Q\}$ no es válida, alcanza con dar valores de las variables que satisfacen P, y tales que luego de ejecutar S, el estado final no satisface Q.

Ejercicio 5. Considere la siguiente especificación de la función `sumarPosicionesImpares`.

```
proc sumarPosicionesImpares (in s: seq⟨ℤ⟩, out result: ℤ) {
      Pre {True}
      Post {result = Σ_{i=0}^{|s|-1} (if i mod 2 = 1 then s[i] else 0 fi)}
}
```

a) Implementar un programa en SmallLang para resolver este problema, que incluya exactamente un ciclo con el siguiente invariante:
$I \equiv 0 \leq j \leq |s| \wedge_L result = \sum_{i=0}^{j-1} (\text{if } i \ mod\ 2 = 1 \text{ then } s[i] \text{ else } 0 \text{ fi})$

b) Demostrar formalmente la corrección del ciclo propuesto.

Ejercicio 6. Considere la siguiente especificación e implementación del problema `maximo`.

Especificación

```
proc maximo (in s: seq⟨ℤ⟩, out i: ℤ) {
      Pre {|s| ≥ 1}
      Post {0 ≤ i < |s| ∧_L
      (∀j :ℤ)(0 ≤ j < |s| →_L s[j] ≤ s[i])}
}
```

Implementación en SmallLang

```
i := 0;
j := 1;
while (j < s.size()) do
  if (s[j] > s[i])
    i := j
  else
    skip
  endif;
  j:=j+1
endwhile
```

a) Escribir la precondición y la postcondición del ciclo.

b) Demostrar que el ciclo es parcialmente correcto, utilizando el siguiente invariante:

$$I \equiv (0 \leq i < |s| \wedge 1 \leq j \leq |s|) \wedge_L (\forall k : \mathbb{Z})(0 \leq k < j \longrightarrow_L s[k] \leq s[i])$$

c) Proponer una función variante que permita demostrar que el ciclo termina.

Ejercicio 7. ★ Considere la siguiente especificación e implementación del problema `copiarSecuencia`.

Especificación

```
proc copiarSecuencia (in s: seq⟨ℤ⟩, inout r: seq⟨ℤ⟩) {
      Pre {|s| = |r| ∧ r = r_0}
      Post {|s| = |r| ∧_L (∀j :ℤ)(0 ≤ j < |s| →_L s[j] = r[j])}
}
```

Implementación en SmallLang

```
i := 0;
while (i < s.size()) do
  r[i]:=s[i];
  i:=i+1
endwhile
```

a) Escribir la precondición y la postcondición del ciclo.

b) Proponer un invariante y demostrar que el ciclo es parcialmente correcto.

c) Proponer una función variante que permita demostrar que el ciclo termina.

Ejercicio 8. Considere la siguiente especificación e implementación del problema `llenarSecuencia`.

Especificación

```
proc llenarSecuencia (inout s: seq⟨ℤ⟩, in d: ℤ, in e: ℤ) {
      Pre {d ≥ 0 ∧ d < |s| ∧ s = S_0}
      Post {|s| = |S_0| ∧_L
      (∀j :ℤ)(0 ≤ j < d →_L s[j] = S_0[j]) ∧
      (∀j :ℤ)(d ≤ j < |s| →_L s[j] = e)}
}
```

Implementación en SmallLang

```
i := d;
while (i < s.size()) do
  s[i]:=e;
  i:=i+1
endwhile
```

a) Escribir la precondición y la postcondición del ciclo.

b) Proponer un invariante y demostrar que el ciclo es parcialmente correcto.

c) Proponer una función variante que permita demostrar que el ciclo termina.

Ejercicio 9. ★ Sea el siguiente ciclo con su correspondiente precondición y postcondición:

```
while (i >= length(s) / 2) do
  suma := suma + s[length(s)-1-i];
  i := i - 1
endwhile
```

$$P_c : \{|s| \ mod \ 2 = 0 \wedge i = |s| - 1 \wedge suma = 0\}$$

$$Q_c : \{|s| \ mod \ 2 = 0 \wedge i = |s|/2 - 1 \ \wedge_L \ suma = \sum_{j=0}^{|s|/2-1} s[j]\}$$

a) Especificar un invariante de ciclo que permita demostrar que el ciclo cumple la postcondición.

b) Especificar una función variante que permita demostrar que el ciclo termina.

c) Demostrar formalmente la corrección y terminación del ciclo usando el Teorema del invariante.

Ejercicio 10. Considere la siguiente especificación del problema `reemplazarTodos`.

```
proc reemplazarTodos (inout s: seq⟨ℤ⟩, in a: ℤ, in b: ℤ) {
      Pre {s = S_0}
      Post {|s| = |S_0| ∧_L
      (∀j :ℤ)((0 ≤ j < |s| ∧_L S_0[j] = a) →_L s[j] = b) ∧
      (∀j :ℤ)(0 ≤ j < |s| ∧_L S_0[j] ≠ a) →_L s[j] = S_0[j])}
}
```

a) Dar un programa en SmallLang que implemente la especificación dada.

b) Escribir la precondición y la postcondición del ciclo.

c) Proponer un invariante y demostrar que el ciclo es parcialmente correcto.

d) Proponer una función variante que permita demostrar que el ciclo termina.

Demostración de correctitud: programas completos

Ejercicio 11. ★ Demostrar que el siguiente programa es **correcto** respecto a la especificación dada.

Especificación

```
proc indice (in s: seq⟨ℤ⟩, in e: ℤ, out r: ℤ) {
      Pre {True}
      Post {r = -1 →
      (∀j :ℤ)(0 ≤ j < |s| →_L s[j] ≠ e)
      ∧
      r ≠ -1 →
      (0 ≤ r < |s| ∧_L s[r] = e)}
}
```

Implementación en SmallLang

```
i := s.size()-1;
j := -1;
while (i>=0) do
  if (s[i]=e) then
    j:=i
  else
    skip
  endif;
  i:=i-1
endwhile;
r := j;
```

Ejercicio 12. ★ Demostrar que el siguiente programa es correcto respecto a la especificación dada.

Especificación

```
proc existeElemento (in s: seq⟨ℤ⟩, in e: ℤ, out r: Bool) {
      Pre {True}
      Post {r = true ↔
      ((∃k :ℤ)(0 ≤ k < |s|) ∧_L s[k] = e)}
}
```

Implementación en SmallLang

```
i := 0;
j := -1;
while (i < s.size()) do
  if (s[i] = e) then
    j := i
 else
   skip
 endif;
 i := i + 1
endwhile;
if (j != -1)
  r := true
else
  r := false
endif
```

Ejercicio 13. Demostrar que el siguiente programa es correcto respecto a la especificación dada.

Especificación

```
proc esSimetrico (in s: seq⟨ℤ⟩, out r:Bool) {
      Pre {True}
      Post {r = true ↔ (∀i : ℤ)(0 ≤ i < |s| →_L
      s[i] = s[|s| − (i + 1)])}
}
```

Implementación en SmallLang

```
i := 0;
j := s.size() - 1;
r := true;
while (i < s.size()) do
  if (s[i]!=s[j]) then
    r := false
  else
    skip
  endif;
  i := i + 1;
  j := j - 1;
endwhile
```

Ejercicio 14. ★ Demostrar que el siguiente programa es correcto respecto a la especificación dada.

Especificación

```
proc concatenarSecuencias (in a: seq⟨ℤ⟩,
in b: seq⟨ℤ⟩,
inout r:seq⟨ℤ⟩) {
      Pre {|r| = |a| + |b| ∧ r = R_0}
      Post {|r| = |R_0| ∧ (∀j :ℤ)(0 ≤ j < |a| →_L r[j] = a[j]) ∧
      (∀j :ℤ)(0 ≤ j < |b| →_L r[j + |a|] = b[j])}
}
```

Implementación en SmallLang

```
i := 0;
while (i < a.size()) do
  r[i]:=a[i];
  i:=i+1
endwhile;
i:= 0;
while (i < b.size()) do
  r[a.size()+i]:=b[i];
  i:=i+1
endwhile
```

Ejercicio 15. Dar dos programas en SmallLang que satisfagan la siguiente especificación, y demostrar que ambos son correctos.

```
proc buscarPosicionUltimoMaximo (in s: seq⟨ℤ⟩, out r:ℤ) {
      Pre {|s| > 0}
      Post {0 ≤ r < |s| ∧_L
      (∀j :ℤ)(0 ≤ j < r →_L s[r] ≥ s[j]) ∧
      (∀j :ℤ)(r < j < |s| →_L s[r] > s[j])}
}
```

Algoritmos y Estructuras de Datos I

Primer Cuatrimestre 2022

Guía Práctica 6
Testing

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1. ★ Sea el siguiente programa:

```
int max(int x, int y) {
L1:  int result = 0;
L2:  if (x<y) {
L3:    result = y;
     } else {
L4:    result = x;
     }
L5: return result;
}
```

Y los siguientes casos de test:

- test1:
  - Entrada x = 0, y=0
  - Resultado esperado result=0
- test2:
  - Entrada x = 0, y=1
  - Resultado esperado result=1

1. Describir el diagrama de control de flujo (control-flow graph) del programa `max`.
2. Detallar qué líneas del programa cubre cada test

Test	L1	L2	L3	L4	L5
test1					
test2					

3. Detallar qué decisiones (branches) del programa cubre cada test

Test	L2-True	L2-False
test1		
test2		

4. Decidir si la siguiente afirmación es verdadera o falsa: *"El test suite compuesto por test1 y test2 cubre el 100 % de las líneas del programa y el 100 % de las decisiones (branches) del programa"*

Ejercicio 2. ★ Sea la siguiente especificación del problema de retornar el mínimo elemento entre dos números enteros:

`proc min` (in x: $\mathbb{Z}$, in y: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
`Pre` $\{True\}$
`Post` $\{(x < y \rightarrow result = x) \wedge (x \geq y \rightarrow result = y)\}$
}

Un programador ha escrito el siguiente programa para implementar la especificación descripta:

```
int min(int x, int y) {
L1:  int result = 0;
L2:  if (x<y) {
L3:    result = x;
     } else {
L4:    result = x;
     }
L5: return result;
}
```

Y el siguiente conjunto de casos de test (test suite):

- minA:
  - Entrada x=1,y=0
  - Salida esperada 0
- minA:
  - Entrada x=0,y=1
  - Salida esperada 0

1. Describir el diagrama de control de flujo (control-flow graph) del programa `min`.
2. ¿La ejecución del test suite resulta en la ejecución de todas las líneas del programa `min`?
3. ¿La ejecución del test suite resulta en la ejecución de todas las decisiones (branches) del programa?
4. ¿Es el test suite capaz de detectar el defecto de la implementación del problema de encontrar el mínimo?
5. En caso de ser necesario, agregar nuevos casos de tests y/o modificar casos de tests existentes para que el test suite detecte el defecto.

Ejercicio 3. ★ Sea la siguiente especificación del problema de sumar y una posible implementación en lenguaje imperativo.

```
proc sumar (in x: ℤ, in y: ℤ, out result: ℤ) {
      Pre {True}
      Post {result = x + y}
}
```

```
int sumar(int x, int y) {
L1:  int result = 0;
L2:  result = result + x;
L3:  result = result + y;
L4:  return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `sumar`.
2. Escribir un conjunto de casos de test (o "test suite") que ejecute todas las líneas del programa `sumar`.

Ejercicio 4. Sea la siguiente especificación del problema de restar y una posible implementación en lenguaje imperativo:

```
proc restar (in x: ℤ, in y: ℤ, out result: ℤ) {
      Pre {True}
      Post {result = x − y}
}
```

```
int restar(int x, int y) {
L1:  int result = 0;
L2:  result = result + x;
L3:  result = result + y;
L4:  return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `restar`.
2. Escribir un conjunto de casos de test (o "test suite") que ejecute todas las líneas del programa `restar`.
3. La línea L3 del programa `restar` tiene un defecto, ¿es el test suite descripto en el punto anterior capaz de detectarlo? En caso contrario, modificar o agregar nuevos casos de test hasta lograr detectarlo.

Ejercicio 5. Sea la siguiente especificación del problema de `signo` y una posible implementación en lenguaje imperativo:

```
proc signo (in x: ℝ, out result: ℤ) {
      Pre {True}
      Post {(result = 0 ∧ x = 0) ∨ (result = −1 ∧ x < 0) ∨ (result = 1 ∧ x > 0)}
}
```

```
int signo(float x) {
L1:  int result = 0;
L2:  if (x<0) {
L3:    result = −1;
L4:  } else if (x>0){
L5:    result = 1;
     }
L6:  return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `signo`.
2. Escribir un test suite que ejecute todas las líneas del programa `signo`.
3. ¿El test suite del punto anterior ejecuta todas las posibles decisiones ("branches") del programa?

Ejercicio 6. Sea la siguiente especificación del problema de `signo` y una posible implementación en lenguaje imperativo:

```
proc signo (in x: ℝ, out result: ℤ) {
      Pre {True}
      Post {(r = 0 ∧ x = 0) ∨ (r = −1 ∧ x < 0) ∨ (r = 1 ∧ x > 0)}
}
```

```
int signo(float x) {
L1:  int result = 0;
L2:  if (x<0) {
L3:    result = −1;
     }
L4:  return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `signo`.
2. Escribir un test suite que ejecute todas las líneas del programa `signo`.
3. Escribir un test suite que ejecute todas las posibles decisiones ("branches") del programa.
4. Escribir un test suite que ejecute todas las líneas del programa pero no ejecute todos las decisiones del programa.

Ejercicio 7. ★ Sea la siguiente especificación:

```
proc fabs (in x: ℤ, out result: ℤ) {
      Pre {True}
      Post {result = |x|}
}
```

Y la siguiente implementación:

```
int fabs(int x) {
L1:   if (x<0 ) {
L2:     return -x;
      } else {
L3:     return x;
  }
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `fabs`.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

Ejercicio 8. ★ Sea la siguiente especificación del problema de `mult10` y una posible implementación en lenguaje imperativo:

```
proc mult10 (in x: Z, out result: Z) {
      Pre {True}
      Post {result = x * 10}
}
```

```
int mult10(int x) {
L1:   int result = 0;
L2:   int count = 0;
L3:   while (count<10) {
L4:       result = result + x;
L5:       count = count + 1;
      }
L6:   return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `mult10`.
2. Escribir un test suite que ejecute todas las líneas del programa `mult10`.
3. ¿El test suite anterior ejecuta todas las posibles decisiones ("branches") del programa?

Ejercicio 9. Sea la siguiente especificación del problema de `sumar` y una posible implementación en lenguaje imperativo:

```
proc sumar (in x: Z, in y: Z, out result: Z) {
      Pre {True}
      Post {result = x + y}
}
```

```
int sumar(int x, int y) {
L1:   int sumando = 0;
L2:   int abs_y = 0;
L3:   if (y<0) {
L4:     sumando = -1;
L5:       abs_y = -y;
      } else {
L7:     sumando = 1;
L8:     abs_y = y;
      }
L9:   int result = x;
L10:  int count = 0;
L11:  while (count < abs_y) {
L12:      result = result + sumando;
L13:      count = count + 1;
      }
L14:  return result;
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `sumar`.
2. Escribir un test suite que ejecute todas las líneas del programa `sumar`.
3. Escribir un test suite que ejecute todas las posibles decisiones ("branches") del programa.

Ejercicio 10. Sea el siguiente programa que computa el máximo común divisor entre dos enteros.

```
int mcd(int x, int y) {
L1:  assert(x >= 0 && y >= 0) // Pre: x e y tienen que ser no negativos
L2:  int tmp = 0;
L3:  while(y != 0) {
L4:    tmp = x %y;
L5:    x = y;
L6:    y = tmp;
  }
L7:  return x; // gcd
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `mcd`.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

Ejercicio 11. ★ Sea el siguiente programa que retorna diferentes valores dependiendo si a, b y c, definen lados de un triángulo inválido, equilátero, isósceles o escaleno.

```
int triangle(int a, int b, int c) {
L1:    if (a <= 0 || b <= 0 || c <= 0) {
L2:        return 4; // invalido
    }
L3:    if (! (a + b > c && a + c > b && b + c > a)) {
L4:        return 4; // invalido
    }
L5:    if (a == b && b == c) {
L6:        return 1; // equilatero
    }
L7:    if (a == b || b == c || a == c) {
L8:        return 2; // isosceles
    }
L9:    return 3;  // escaleno
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `triangle`.
2. Escribir un test suite que ejecute todas las líneas y todos los branches del programa.

Ejercicio 12. ★ Sea la siguiente especificación del problema de `multByAbs` y una posible implementación en lenguaje imperativo:

`proc multByAbs` (in x: $\mathbb{Z}$, in y:$\mathbb{Z}$, out result: $\mathbb{Z}$) {
  `Pre` $\{True\}$
  `Post` $\{result = x * |y|\}$
}

```
int multByAbs(int x, int y) {
L1:  int abs_y = fabs(y); // ejercicio anterior
L2:  if (abs_y<0) {
L3:    return -1;
  } else {
L4:    int result = 0;
```

```
L5:     int i = 0;
L6:     while (i<abs_y) {
L7:       result = result + x;
L8:       i = i + 1;
        }
L9:     return result;
    }
}
```

1. Describir el diagrama de control de flujo (control-flow graph) del programa `multByAbs`.
2. Detallar qué líneas y branches del programa no pueden ser cubiertos por ningún caso de test. ¿A qué se debe?
3. Escribir el test suite que cubra todas las líneas y branches que puedan ser cubiertos.

Ejercicio 13. Sea la siguiente especificación del problema de `vaciarSecuencia` y una posible implementación en lenguaje imperativo:

```
proc vaciarSecuencia (inout s: seq⟨Z⟩) {
      Pre {S_0 = s}
      Post {|s| = |S_0| ∧
      (∀j : Z)(0 ≤ j < |s| →_L s[j] = 0)}
}
```

```
void vaciarSecuencia(vector<int> &s) {
L1,L2,L3:  for (int i=0; i<s.size();i++) {
L4:            s[i]=0;
           }
}
```

1. Escribir el diagrama de control de flujo (control-flow graph) del programa `vaciarSecuencia`.
2. Escribir un test suite que cubra todos las líneas de programa (observar que un `for` contiene 3 líneas distintas)
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 14. Sea la siguiente especificación del problema de `existeElemento` y una posible implementación en lenguaje imperativo:

```
proc existeElemento (in s: seq⟨Z⟩, in e:Z, out result: Bool) {
      Pre {True}
      Post {result = True ↔ (∃j : Z)(0 ≤ j < |s| ∧_L s[j] = e)}
}
```

```
bool existeElemento(vector<int> s, int e) {
L1:         bool result = false;
L2,L3,L4:   for (int i=0; i<s.size();i++) {
L5:           if (s[i]==e) {
L6:             result = true;
L7:             break;
              }
            }
L8:           return result;
}
```

1. Escribir el diagrama de control de flujo (control-flow graph) del programa `existeElemento`.
2. Escribir un test suite que cubra todos las líneas de programa (observar que un `for` contiene 3 líneas distintas)

3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 15. Sea la siguiente especificación del problema de `cantidadDePrimos` y una posible implementación en lenguaje imperativo:

proc `cantidadDePrimos` (in n: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
  Pre $\{n \geq 0\}$
  Post $\{result = \sum_{i=2}^{i \leq n} \big(\text{if } ((\forall j : \mathbb{Z})((1 < j < i) \rightarrow i \bmod j \neq 0)) \text{ then 1 else 0 fi}\big)\}$
}

```
int cantidadDePrimos(int n) {
L1:          int result = 0;
L2,L3,L4:    for (int i=2; i<=n;i++) {
L5:            bool inc = esPrimo(i);
L6:            if (inc==true) {
L7:              result++;
               }
             }
L8:          return result;
}
/* procedimiento auxiliar */
          bool esPrimo(int x) {
L9:            int result = true;
L10,L11,L12:   for (int i=2 ; i <x; i++) {
L13:             if (x %i == 0) {
L14:               result = false;
L15:               break;
               }
             }
L16:           return result;
}
```

1. Escribir los diagramas de control de flujo (control-flow graph) para `cantidadDePrimos` y la función auxiliar `esPrimo`.
2. Escribir un test suite que cubra todos las líneas de programa del programa `cantidadDePrimos` y `esPrimo`.
3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Ejercicio 16. Sea la siguiente especificación del problema de `esSubsecuencia` y una posible implementación en lenguaje imperativo:

proc `esSubsecuencia` (in s: $seq\langle\mathbb{Z}\rangle$, in r: $seq\langle\mathbb{Z}\rangle$,out result: Bool) {
  Pre $\{True\}$
  Post $\{result = True \leftrightarrow |r| \leq |s|$
  $\wedge_L (\exists i : \mathbb{Z})((0 \leq i < |s| \wedge i + |r| < |s|) \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |r| \rightarrow_L s[i + j] = r[j]))\}$
}

```
1  bool esSubsecuencia(vector<int> s, vector<int> r) {
2    bool result = false;
3    int ultimoIndice = s.size() - r.size();
4    for (int i = 0; i< ultimoIndice; i++) {
5
6      /* obtener una subsecuencia de s */
7      vector<int> subseq = subsecuencia(s,i,r.size());
8
9      /* chequear si lasubsecuencia es igual a  r */
10     bool sonIguales = iguales(subseq, r);
```

```
11      if (sonIguales==true) {
12        result = true;
13        break;
14      }
15    }
16    return result;
17  }
18  /* procedimiento auxiliar subsecuencia*/
19  vector<int> subsecuencia(vector<int> s, int desde, int longitud) {
20    vector<int> rv;
21    int hasta = desde+longitud;
22    for (int i=desde;i<hasta ;i++) {
23      int elem = s[i];
24      rv.push_back(elem);
25    }
26    return rv;
27  }
28  /* procedimiento auxiliar iguales*/
29  bool iguales(vector<int> a, vector<int> b) {
30    bool result = true;
31    if (a.size()==b.size()) {
32      for (int i=0;i<a.size() ;i++) {
33        if (a[i]!=b[i]) {
34          result = false;
35          break;
36        }
37      }
38    } else {
39      result = false;
40    }
41    return result;
42  }
```

1. Escribir los diagramas de control de flujo (control-flow graph) para `esSubsecuencia` y las funcones auxiliares `subsecuencia` e `iguales`.

2. Escribir un test suite que cubra todos las líneas de programa *ejecutables* de todos los procedimientos. Observar que un `for` contiene 3 líneas distintas.

3. En caso que el test suite del punto anterior no cubriera todo los branches del programa, extenderlo de modo que logre cubrirlos.

Algoritmos y Estructuras de Datos I
Primer Cuatrimestre 2022

Guía Práctica 7
Ciclos a partir de invariantes

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1.
Sea la siguiente especificación del problema de copiar una secuencia de enteros:

`proc copiarSecuencia` (in s: $seq\langle\mathbb{Z}\rangle$, out result: $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{True\}$
`Post` $\{s = result\}$
}

Sea la siguiente implementación incompleta de la función copiarSecuencia:

```
vector<int> copiarSecuencia(vector<int> s) {
    vector<int> r;
    int i = 0;
    while(i<s.size()) {
      ...
    }
    return r;
  }
}
```

Completar el programa (i.e. escribir el cuerpo del `while`) de forma que cumpla el siguiente invariante de ciclo:

$$I = (0 \leq i \leq |s| \wedge |r| = i) \wedge_L (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = r[j])$$

Ejercicio 2.
Sea la siguiente especificación:

`proc incSecuencia` (inout s: $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{s = S_0\}$
`Post` $\{|s| = |S_0| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[i] = S_0[i] + 1)\}$
}

Sea la siguiente implementación incompleta de la función incSecuencia:

```
void incSecuencia(vector<int> &a) {
    int i = 0;
    while(...) {
      ...
    }
}
```

Completar el programa (i.e. escribir el cuerpo del `while` y su guarda) de forma que cumpla el siguiente invariante de ciclo:

$$I = 0 \leq i \leq |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L s[j] = S_0[j] + 1)$$

Ejercicio 3.
Sea la siguiente especificación del problema de retornar la cantidad de apariciones de un elemento en una secuencia de enteros:

`proc cantApariciones` (in s: $seq\langle\mathbb{Z}\rangle$, in e: $\mathbb{Z}$, out result: $\mathbb{Z}$) {
`Pre` $\{True\}$
`Post` $\{result = \#apariciones(s, e))\}$
}

Sea la siguiente implementación incompleta de la función cantApariciones:

```
int cantApariciones(vector<int> s, int e) {
    int r = 0;
    for(int i=0; ... ; ...) {
      ...
    }
    return r;
}
```

Completar el programa (i.e. escribir el cuerpo y la declaración del `for`) de forma que cumpla el siguiente invariante de ciclo:

$$I = 0 \leq i \leq |s| \wedge_L r = \#apariciones(subseq(s, 0, i), e))$$

Ejercicio 4.
Sea la siguiente especificación de un ciclo:

- $P_c : i = -1 \wedge s = S_0$
- $Q_c : |s| = |S_0| \wedge_L (\forall z : \mathbb{Z})(0 \leq z < |s| \rightarrow_L s[z] = S_0[z]^2)$

Dar dos implementaciones distintas que satisfagan la especificación del ciclo con el siguiente invariante:

$$I = (|s| = |S_0| \wedge -1 \leq i \leq |s| - 1) \wedge_L (\forall j : \mathbb{Z})(0 \leq j \leq i \rightarrow_L s[j] = S_0[j]^2) \wedge (\forall j : \mathbb{Z})(i < j < |s| \rightarrow_L s[j] = S_0[j])$$

Ejercicio 5.
Escribir un programa para el siguiente problema que respete la especificación y el invariante dado

`proc duplicarElementos` (inout s : $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{s = s_0 \wedge |s| \ mod \ 2 = 0\}$
`Post` $\{|s| = |s_0| \wedge_L$
$(\forall i : \mathbb{Z})(enRango(i, s) \longrightarrow_L s[i] = 2 * s_0[i])\}$

$I \equiv (|s| = |s_0| \wedge (0 \leq i \leq |s|/2)) \wedge_L$
$subseq(s, 0, |s| - 2 * i) = subseq(s_0, 0, |s_0| - 2 * i) \wedge$
$(\forall k : \mathbb{Z})(|s| - 2 * i \leq k < |s| \longrightarrow_L s[k] = 2 * s_0[k]))$

Ejercicio 6.
Escribir un programa para el siguiente problema que respete la especificación y contenga un ciclo el invariante dado

`proc dividirPorPromedio` (inout s : $seq\langle\mathbf{R}\rangle$) {
`Pre` $\{s = s_0 \wedge |s| \ mod \ 2 = 0 \wedge |s| > 0\}$
`Post` $\{|s| = |s_0| \wedge_L$
$(\forall i : \mathbb{Z})(enRango(i, s) \longrightarrow_L s[i] = \frac{s_0[i]}{promedio(s_0)})\}$
`aux promedio` (s : $seq\langle\mathbf{R}\rangle$) : $\mathbf{R} = \frac{\sum_{i=0}^{|s|-1} s[i]}{|s|}$;

$I \equiv (|s| = |s_0| \wedge 0 \leq i \leq \frac{|s|}{2}) \wedge_L$
$subseq(s, i, |s| - i) = subseq(s_0, i, |s_0| - i) \wedge$
$(\forall k : \mathbb{Z})(0 \leq k < i \longrightarrow_L s[k] = \frac{s_0[k]}{promedio(s_0)}) \wedge$
$(\forall k : \mathbb{Z})(|s| - i - 1 < k < |s| \longrightarrow_L s[k] = \frac{s_0[k]}{promedio(s_0)})$

Ejercicio 7.
Dar un programa que satisfaga la especificación y tenga un ciclo con el invariante:

`proc armarPiramide` (in v: $\mathbb{Z}$, inout l: $seq\langle\mathbb{Z}\rangle$) {
`Pre` $\{l = L_0\}$
`Post` $\{|L_0| = |l| \wedge esPiramide(l, v)\}$
`pred esPiramide` (l: $seq\langle\mathbb{Z}\rangle$, v: $\mathbb{Z}$) {

$(\forall j : \mathbb{Z})(0 \leq j < |l|/2 \Rightarrow_L l[j] = v + j) \wedge$
$(\forall j : \mathbb{Z})(|l|/2 \leq j < |l| \Rightarrow_L l[j] = v + |l| - j - 1)$

}
}

a. **Invariante:** $|l| = |L_0| \wedge |l|/2 \leq i \leq |l| \wedge_L ((i = |l|/2 \wedge l = L_0) \vee_L$
$(\exists p : seq\langle\mathbb{Z}\rangle)(esPiramide(p, v) \wedge |p| = |l| \wedge_L subseq(p, |l| - i, i) = subseq(l, |l| - i, i)))$

b. **Invariante:** $|l| = |L_0| \wedge 0 \leq i \leq |l| \wedge_L piramideHastaI(l, i, v)$
`pred piramideHastaI` (l: $seq\langle\mathbb{Z}\rangle$, i: $\mathbb{Z}$, v: $\mathbb{Z}$) {
$(\exists p : seq\langle\mathbb{Z}\rangle)esPiramide(p, v) \wedge |p| = |l| \wedge_L subseq(p, 0, i) = subseq(l, 0, i)$
}

Ejercicio 8.
Escribir un programa para el siguiente problema que respete la especificación y el invariante dado

`proc multiplicar` (inout s : $seq\langle\mathbb{R}\rangle$) {
`Pre` $\{s = s_0 \wedge |s| \ mod \ 4 = 0\}$
`Post` $\{|s| = |s_0| \wedge_L$
$(\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L s[i] = 10 * s_0[i])\}$
}

$I \equiv (|s| = |s_0| \wedge \frac{|s|}{2} \leq i \leq |s|) \wedge_L esPar(i) \wedge$
$subseq(s, i, |s|) = subseq(s_0, i, |s_0|) \wedge subseq(s, 0, |s| - i) = subseq(s_0, 0, |s_0| - i) \wedge$
$(\forall k : \mathbb{Z})(\frac{|s|}{2} \leq k < i \longrightarrow_L s[k] = 10 * s_0[k]) \wedge (\forall k : \mathbb{Z})(|s| - i \leq k < \frac{|s|}{2} \longrightarrow_L s[k] = 10 * s_0[k])$

Ejercicio 9.
Escribir un programa para el siguiente problema que respete la especificación y el invariante dado

`proc cerearYsumar` (inout s : $seq\langle\mathbb{Z}\rangle$, inout suma : $\mathbb{Z}$) {

`Pre` $\{s = s_0 \wedge |s| \ mod \ 8 = 0\}$

`Post` $\{|s| = |s_0| \wedge_L ((\forall i : \mathbb{Z})(enRango(i, s) \longrightarrow_L s[i] = 0) \wedge suma = \sum_{i=0}^{|s|-1} s_0[i])\}$

}

$I \equiv (|s| = |s_0| \wedge 0 \leq i \leq |s|/4) \wedge_L$
$(subseq(s, 2 * i, |s| - 2 * i) = subseq(s_0, 2 * i, |s_0| - 2 * i) \wedge (\forall k : \mathbb{Z})(0 \leq k < 2 * i \longrightarrow_L s[k] = 0) \wedge$
$(\forall k : \mathbb{Z})(|s| - 2 * i \leq k < |s| \longrightarrow_L s[k] = 0) \wedge suma = \sum_{j=0}^{2*i-1} s_0[j] + \sum_{j=|s|-2*i}^{|s|-1} s_0[j])$

Algoritmos y Estructuras de Datos I

Primer Cuatrimestre 2022

Guía Práctica 8

Tiempo de Ejecución de Peor Caso de un Programa

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1.
Contar la cantidad de operaciones elementales que realizan los siguientes programas.

```
int ultimo1(vector<int>& v) {
  return v[v.size() - 1];
}

int ultimo2(vector<int>& v) {
  int i = v.size();
  return v[i - 1];
}

int ultimo3(vector<int>& v) {
  int i = 0;
  while (i < v.size()) {
    i++;
  }
  return v[i - 1];
}
```

Ejercicio 2.
Calcular el tiempo de ejecución de peor caso (en notación O grande) de los siguientes programas con respecto al tamaño de los secuencias de entrada. Recordar que tanto la lectura como la escritura de un elemento en un vector tiene tiempo de ejecución perteneciente a $O(1)$.

```
void f1(vector<int> &vec){
  i = vec.size() / 2;
  while( i >= 0){
    vec[vec.size() / 2 - i] = i;
    vec[vec.size() / 2 + i] = i;
    i--;
  }
}

// pre: |vec| > 20000
void f2(vector<int> &vec){
  i = 0;
  while(i < 10000){
    vec[vec.size() / 2 - i] = i;
    vec[vec.size() / 2 + i] = i;
    i++;
  }
}

// pre: e pertenece a v1
int f3(vector<int> &v1, int e){
  int i = 0;
  while (v1[i] != e){
    i++;
  }
  return i ;
}
```

```
void f4(vector<int> &vec){
  int rec = 0;
  int max_iter = 1000;
  if max_iter > vec.size(){
    max_iter = vec.size();
  }

  for(int i=0; i < max_iter; i++){
    for(int j=0; j < max_iter; j++){
      res += vec[i] * vec[j];
    }
  }
}

void f5(vector<int> &v1, vector<int> &v2){
  vector<int> res(v1.size()+v2.size(),0);
  // inicializa vector en O(|a|+|b|)
  for(int i=0; i < v1.size(); i ++){
    res[i]=v1[i]; // O(1)
  }
  for(int i=0; i < v2.size(); i ++){
    res[v1.size()+i]=v2[i]; // O(1)
  }
  return;
}
```

Ejercicio 3.
Verdadero o falso (justificar).

a) A y B son dos programas que resuelven el mismo problema con un vector v como única entrada. Para un vector de tamaño 100 A demora 2 minutos en ejecutarse y bajo las mismas condiciones (misma entrada, computadora, recursos, etc) B demora 20 segundos. Por lo tanto, el programa B es más eficiente.

b) Si el tiempo de ejecución de peor caso de dos programas pertenece a $O(n)$, entonces el tiempo de cómputo es equivalente (variando un poco según el estado de la computadora en el momento de ejecutar).

c) Se tienen dos programas cuyo tiempo de ejecución de peor caso perteneciente a $O(n)$ y a $O(log(n))$ respectivamente. Para cualquier entrada con tamaño mayor a 100, el segundo programa demorará menos tiempo en ejecutar.

d) Se tienen dos programas con tiempo de ejecución de peor caso perteneciente a $O(n)$ y a $O(log(n))$ respectivamente. Para cualquier entrada con tamaño mayor a un cierto número, el segundo programa demorará menos tiempo en ejecutar.

e) Si se hace un llamado a una función cuyo tiempo de ejecución de peor caso pertenece a $O(n^2)$ dentro de un ciclo, el tiempo de ejecución de peor del ciclo resultante pertenecerá a $O(n^3)$.

Ejercicio 4.

```
int mesetaMasLarga(vector<int> &v) {
  int i = 0;
  int maxMeseta = 0;
  int meseta;
  while (i < v.size()) {
    int j = i + 1;
    while (j < v.size() && v[i] == v[j]) {
      j++;
    }
    meseta = j - i;
    i = j;

    if (meseta > maxMeseta) {
      maxMeseta = meseta;
    }
  }
  return maxMeseta;
}
```

a) ¿Qué hace este programa?

b) Calcular el tiempo de ejecución de peor caso de este programa en función del tamaño del vector.

c) ¿Es posible escribir otro programa que resuelva el problema utilizando solo un ciclo?

Ejercicio 5.

```
vector<int> hacerAlgo(vector<int> &v) {
  vector<int> res;
  for (int i = 0; i < 100; i++) {
    res.push_back(contarApariciones(v, i+1));
  }

  return res; // copia el vector
}

int contarApariciones(vector<int> &v, int elem) {
  int cantAp = 0;
  for (int i = 0; i < v.size(); i++) {
  if (v[i] == elem) {
      cantAp++;
```

```
    }
  }
}
```

Calcular el tiempo de ejecución de peor caso del programa `hacerAlgo` con respecto a $|v|$.

Ejercicio 6.
Dadas las siguientes funciones:

```
bool f(vector<float> s) {
    float p;
    float res = 0;
    for (int i = 0; i < s.size(); i++) {
        p = g(h(s, i)));
        if (p > res) {
            res = p;
        }
    }
    return res;
}

float g(vector<float> v) {
    float p = 1;
    for (int i = 0; i < v.size(); i++) {
        p = p * v[i];
    }
    return p;
}
```

```
// Pre: n <= v.size()
vector<float> h(vector<float> w, int n) {
    vector<float> res;
    int i = 0;
    while (i < n && w[i] > 0) {
        res.push_back(w[i]);
        i++;
    }
    return res;
}
```

1. Explicar brevemente con sus palabras qué cómputo realiza la función f. Hacerlo en a lo sumo un párrafo acompañado de un ejemplo (no explicar cómo lo hace).
2. Indicar el tiempo de ejecución de peor caso de h en función del n. Justificar.
3. Dar un ejemplo del peor caso y un ejemplo de mejor caso para la función h. Justificar.
4. Indicar el tiempo de ejecución de peor caso de f en función del tamaño de s. Justificar.

Ejercicio 7.

```
int sumarPotenciaHasta(int n) {
  int res = 0;
  int i = 1;
  while(i < n) {
    res = res + i;
    i = i * 2;
  }
  return res;
}
```

a) ¿Qué tiempo de ejecución de peor caso tiene el programa en función del valor n?

b) ¿Es posible calcular la suma de potencias hasta n con un tiempo de ejecución de peor caso asintóticamente mejor que el programa del punto anterior si la pre condición es True?

c) ¿Es posible calcular la suma de potencias hasta n con un tiempo de ejecución de peor caso asintóticamente mejor que el item a si la pre condición es $(\exists x : \mathbb{Z})\ n = 2^x + 1$?

Ejercicio 8.
Una matriz cuadrada se dice *triangular* si todos los elementos por debajo de la diagonal son iguales a cero.

a) Escribir un programa que calcule el determinante de una matriz triangular. Recordar que el determinante de una matriz triangular es el producto de los elementos de su diagonal.

b) Escribir un programa que determine si una matriz de $N \times N$ es o no triangular.

c) Calcular el tiempo de ejecución de peor caso de los programas:

   a) en función de la cantidad de elementos de la matriz.

   b) en función de la cantidad de *filas* de la matriz.

Ejercicio 9.
Dadas dos matrices A y B se desea multiplicarlas siguiendo esta especificación:

proc multiplicar (in m1: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, in m2: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$, out res: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
Pre {*completar*}
Post $\{|res| = |m1| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |m1| \rightarrow_L (|res[i]| = |m2[0]| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |m2[0]| \rightarrow_L res[i][j] = \sum_{k=0}^{|m2|-1} m1[i][k] * m2[k][j])))\}$
}

a) Completar la precondición del problema.

b) Escribir un programa que retorne AB

c) Determinar el tiempo de ejecución de peor caso de este programa en función de:

   a) La cantidad de filas y columnas de cada una de las matrices.

   b) Suponiendo que $N = \text{filas}_{m1} = \text{filas}_{m2} = \text{columnas}_{m1} == \text{columnas}_{m2}$

Ejercicio 10.
Sea *mat* una matriz de $N \times N$ y la siguiente función:

```
void sumasAcumuladas(vector<vector<int>> &mat, int N){
  for(int i = N - 1; i >= 0, i--) {
    for(int j = N - 1, j >= 0; j--) {
      mat[i][j] = sumHasta(mat, i, j, N);
    }
  }
}

int sumHasta(vector<vector<int> &mat, int I, int J, int N) {
  int res = 0;
  int lim;
  for (int i = 0; i <= I; i++) {
    if (i == I) {
      lim = J
    } else {
      lim = N - 1;
    }
    for (int j = 0; j <= lim; j++) {
      res += mat[i][j]
    }
  }
  return res;
}
```

1. Calcular el tiempo de ejecución de peor caso del programa `sumasAcumuladas` en función de la cantidad de elementos de la matriz.
2. Dar un programa que resuelva sumasAcumuladas cuyo tiempo de ejecución en peor caso sea $O(n^2)$

Ejercicio 11. Dada una secuencia de n números enteros, dar un programa que encuentre la máxima cantidad de elementos impares consecutivos cuya tiempo de ejecución de peor caso pertenezca a $O(n)$.

Ejercicio 12. Escribir un programa que sea correcto respecto de la siguiente especificación, y cuyo tiempo de ejecución de peor caso pertenezca a $O(|s|)$.

```
proc restarAcumulado (in s: seq⟨ℤ⟩, in x: ℤ, out res: seq⟨ℤ⟩)  {
      Pre {True}
      Post {|res| = |s| ∧_L (∀i : ℤ)(0 ≤ i < |s| →_L res[i] = x − Σ_{j=0}^{i} s[j])}
}
```

Ejercicio 13. Sea m una matriz de $N \times N$ donde cada posición contiene el costo de recorrer dicha posición (dicho costo es positivo para todas las posiciones). Asumiendo que la única forma de recorrer la matriz es avanzando hacia abajo y hacia la derecha, escribir un programa que calcule el mínimo costo para llegar desde la posición $(1, 1)$ hasta la posición (N, N) y cuyo tiempo de ejecución de peor caso pertenezca a $O(N^2)$.

Ejercicio 14.
Sea A una matriz cuadrada y n un número natural.

a) Escribir un programa que calcule $\prod_{i=1}^{n} A$ (es decir A^n) (reutilizar el programa del ejercicio **??**) ¿Cuál es el tiempo de ejecución de peor caso de esta función?

b) Resolver el punto anterior suponiendo que $n = 2^m$ (n potencia de 2) ¿Se pueden reutilizar cuentas ya realizadas? ¿Cuál es el tiempo de ejecución de peor caso para cada programa?

Ejercicio 15.
Dada una matriz de booleanos de n filas y m columnas con n impar. Se sabe que hay exactamente una fila que no está repetida, y el resto se encuentra exactamente dos veces en la matriz.

a) Escribir un programa que devuelva un vector con los valores de la fila que no se repite. ¿Cuál es su tiempo de ejecución de peor caso descripto ?

b) ¿Es posible un programa que recorra cada casillero de la matriz sólo una vez? En caso de no haberlo resuelto con esta restricción, modificar el programa para que la cumpla. ¿Cuál es su tiempo de ejecución de peor caso ?

c) La solución al punto anterior, ¿utiliza vectores auxiliares?, en caso que lo haga, escribir un programa que no los necesite.